2020

Deep Recurrent Networks for Gesture Recognition and Synthesis

Mehran Maghoumi
University of Central Florida

DEEP RECURRENT NETWORKS FOR GESTURE RECOGNITION AND SYNTHESIS

by

MEHRAN MAGHOUMI
B.S. Computer Engineering, Sheikhbahaee University, 2011
M.S. Computer Science, Brock University, 2014
M.S. Computer Science, University of Central Florida, 2017

A dissertation submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy
in the Department of Computer Science
in the College of Engineering & Computer Science
at the University of Central Florida
Orlando, Florida

Fall Term
2020

Major Professor: Joseph J. LaViola Jr.

ABSTRACT

It is hard to overstate the importance of gesture-based interfaces in many applications nowadays. The adoption of such interfaces stems from the opportunities they create for incorporating natural and fluid user interactions. This highlights the importance of having gesture recognizers that are not only accurate but also easy to adopt. The ever-growing popularity of machine learning has prompted many application developers to integrate automatic methods of recognition into their products. On the one hand, deep learning often tops the list of the most powerful and robust recognizers. These methods have been consistently shown to outperform all other machine learning methods in a variety of tasks. On the other hand, deep networks can be overwhelming to use for a majority of developers, requiring a lot of tuning and tweaking to work as expected. Additionally, these networks are infamous for their requirement for large amounts of training data, further hampering their adoption in scenarios where labeled data is limited.

In this dissertation, we aim to bridge the gap between the power of deep learning methods and their adoption into gesture recognition workflows. To this end, we introduce two deep network models for recognition. These models are similar in spirit, but target different application domains: one is designed for segmented gesture recognition, while the other is suitable for continuous data, tackling segmentation and recognition problems simultaneously. The distinguishing characteristic of these networks is their simplicity, small number of free parameters, and their use of common building blocks that come standard with any modern deep learning framework, making them easy to implement, train and adopt. Through evaluations, we show that our proposed models achieve state-of-the-art results in various recognition tasks and application domains spanning different input devices and interaction modalities. We demonstrate that the infamy of deep networks due to their demand for powerful hardware as well as large amounts of data is an unfair assessment. On the contrary, we show that in the absence of such data, our proposed models can be quickly trained

while achieving competitive recognition accuracy.

Next, we explore the problem of synthetic gesture generation: a measure often taken to address the shortage of labeled data. We extend our proposed recognition models and demonstrate that the same models can be used in a Generative Adversarial Network (GAN) architecture for synthetic gesture generation. Specifically, we show that our original recognizer can be used as the discriminator in such frameworks, while its slightly modified version can act as the gesture generator. We then formulate a novel loss function for our gesture generator, which entirely replaces the need for a discriminator network in our generative model, thereby significantly reducing the complexity of our framework. Through evaluations, we show that our model is able to improve the recognition accuracy of multiple recognizers across a variety of datasets. Through user studies, we additionally show that human evaluators mistake our synthetic samples with the real ones frequently indicating that our synthetic samples are visually realistic.

Additional resources for this dissertation (such as demo videos and public source codes) are available at *https://www.maghoumi.com/dissertation*.

This dissertation is dedicated to all Ph.D. dropouts. Rest assured, you did not miss out on much!

ACKNOWLEDGMENTS

I would like to thank my family Esmaeil, Fatemeh, Maryam, Mojgan, Mahshad, and Alaleh for their unrelenting support and encouragement. This would not have been possible without you. Thanks for believing in me, your emotional support, and sticking with me all the way to the end.

I would also like to thank my advisor Dr. Joseph J. LaViola Jr. for his support and superb supervision throughout this journey. I also wish to thank the esteemed members of my dissertation committee Dr. Charles Hughes, Dr. Gita Sukthankar and Dr. Amir Gholaminejad for their valuable guidance and feedback.

Furthermore, I wish to extend gratitude and appreciation to my friends as well as the members of the ICE lab whose feedback helped me along this path. Namely, I would like to thank Eugene Taranta for brainstorming with me and all the late-night chats. I do not recall text messaging with a single sole more than I did with you! I wish to also thank Corey Pittman for selflessly helping me more times than I can remember. I additionally want to thank my colleagues at NVIDIA for inspiring me to continue this journey: Evan Dong, Davide Onofrio, Ankit Gupta, Younghoon Lee, Art Tevs, and Josh Abbot. Without your advice and guidance this work would be left half-done.

Last but not least, special thanks go to Miguel Sainz for his unwavering support and believing in me, and also for giving me a chance to do what I loved but could only dream of: you are truly a rock star. I made you a personal promise to see this work through. Promise fulfilled!

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

In recent years, gesture-based interfaces have become more relevant than ever. Due to their simplicity and intuitiveness, we observe a trend of integrating gestures into almost every consumer product that supports interaction. From entertainment devices and home appliances to vehicles, we see an ever increasing number of devices that support at least one form of a gestural interface such as touch inputs and mid-air swipes to name but a few.

The commoditization of gesture-based interaction is due in large part to the advent of various input devices. As input devices get more capable and precise, the complexity of the interactions that they can capture also increases. This, in turn, ignites the need for gesture recognition methods that can leverage these capabilities [5].

From a practitioner's point of view, any particular gesture recognizer would need to possess a set of traits in order to gain adoption. On the one hand, it is expected of any recognition method to capture the fine differences among gestures and distinguish one gesture from another with a high degree of confidence. This expectation stems from the fact that the usability of any input device is bounded by the software that works with the data generated by the underlying hardware. On the other hand, a desirable property of gesture recognition methods is their ability to work with a vast number of input devices and modalities (2D pen and touch, 3D hand or full-body gestures, *etc*.). Finally, a recognition method should be *accessible*: a system designer should ideally be able to integrate the method into their workflow with the least amount of effort. This suggests that the recognizer should be suitable for various development stages: from prototyping to production.

Perhaps a few years ago, one would assert that the *flexibility* and the *power* of recognizers are properties which are often at odds. Yet, the advent of deep neural networks has completely transformed the landscape of solving recognition problems. Using nothing but data, we can solve challeng-

ing recognition problems without hand-engineering a single feature, or even understanding what a deep network is computing under the hood! Yet, the magnificent ability of deep networks in solving recognition problem comes at a cost: the need for a huge amount of data. Thus if one puts the need for powerful recognizers on the one side of the scale, the other side would undoubtedly contain the need for the data that the recognizer requires.

While the size of publicly available datasets grows each and every day, obtaining task-specific data is not always easy, requiring a lot of time and care. To take the extreme into consideration, collecting more data is sometimes very costly (such as labeling datasets that require domain experts, as is usually the case with medical data) or even impossible (think of applications which support gesture customization: there is a limit to how many examples a single user can provide).

The difficulty in collecting large sets of training data has fueled the need for synthetic generation methods that can take an existing dataset, and generate new samples from it. Consequently, *data augmentation* has become an indispensable part of modern deep learning research and development. One paradigm for data generation that has recently enjoyed great success and has been established as the de facto in various generation tasks is a group of neural networks dubbed Generative Adversarial Networks (GAN). These networks, which typically consist of a *generator* and a *discriminator*, work by estimating the distribution of the samples of a given dataset. This estimate is improved repeatedly through the adversarial training process formulated as a zero-sum game. The generator generates fake samples, and the discriminator seeks to determine whether a given sample is real or fake. During training, the generator aims to fool the ever-improving discriminator into thinking a fake sample is real. When training concludes, new examples can be sampled from this estimated distribution, yielding realistic samples of the underlying data.

1.1 Statement of Research

Thus far, we have outlined a few seemingly contradicting problems: the need for gestural interfaces has prompted the need for powerful gesture recognizers. These recognizers should work across various devices and modalities, yet they should be accessible, *i.e*. easily understood and implemented to be suitable for various development stages: from prototyping to production. These goals are often at odds: the recognition power of a recognizer usually comes at the cost of increased complexity and decreased flexibility of working across different input devices and modalities. Considering the success of deep networks, and their ability to learn from data, a logical choice of the recognizer seems to be one which is based on deep networks. Yet, deep networks are notorious for their complexity and their need for a lot of data. While generative models are viable choices for generating synthetic data, their development, training and tuning may be complex and time consuming.

In this work, we aim to reconcile these contradicting goals. Specifically, we explore the application of deep network models for two distinct, yet related tasks: gesture recognition and gesture generation. While gesture recognition has received a lot of attention in the literature, we devise a simplified recurrent network that can be used for recognition across a variety of input devices and gesture modalities. Secondary to this, we discuss the development of an accessible model: a model with a low number of free parameters, that does not require much tuning and is quick to train even without powerful hardware.

Additionally, we discuss deep recurrent networks for synthetic gesture generation. Although various generative modeling techniques exist, we specifically focus on GANs as to our knowledge their utilization for generating synthetic gesture sequences remains a largely unexplored problem. We then turn our focus to simplifying GANs and propose a *non-adversarial* gesture generator: a model which is inspired by GANs yet does not require a separate discriminator. We show that our proposed model is significantly faster to train and outperforms the GAN-based gesture generator.

We additionally discuss gesture generation using variational autoencoders (VAE) as well as the use of our proposed gesture generator for domain adaptation tasks.

1.1.1 Thesis Statement

It is possible to devise a single family of deep recurrent neural networks, comprised of only gated recurrent units and fully connected layers, which can be used for segmented gesture recognition, unsegmented gesture recognition as well as synthetic gesture generation, while achieving competitive performance in each problem domain.

1.2 Contributions

With regards to segmented gesture recognition, our main contributions are devising a novel network model that works with raw vector data and is: **(1)** intuitive to understand and easy to implement, **(2)** easy to use, works out-of-the-box on noisy data, and is easy to train, without requiring powerful hardware **(3)** achieves state-of-the-art results in various use-cases, even with a limited amount of training data. We believe **(1)** and **(2)** would make our gesture recognizer enticing for application developers while **(3)** appeals to seasoned practitioners. To our knowledge, no prior work specifically focuses on model simplicity, accessibility for the masses, small training sets or CPU-only training which we think makes our model unique among similar models in the literature. Afterwards, we show that this model can be easily extended to continuous gesture recognition with competitive performance across various datasets.

With regards to synthetic gesture generation with recurrent networks, our main contributions are **(1)** a novel recurrent GAN model for gesture generation that works across a variety of datasets and modalities, **(2)** a novel and intuitive loss function that completely replaces the discriminator of our

GAN model, which not only simplifies and significantly speeds up the training process, but also yields a generator that produces high quality examples, **(3)** an evaluation of the improvements in gesture recognition accuracy when our generator is used for data augmentation. Our evaluations consist of measuring the improvements in gesture recognition accuracy with the use of synthetic data, as well as the perceived realism of the produced samples by human evaluators via a user study.

We additionally discuss the use of our proposed model for domain adaptation tasks and show that our GAN-based model can in fact be used in such application domains. Lastly, we provide the first look into generating synthetic gestures in a variational autoencoder (VAE) setting. We propose a novel formulation of a VAE loss function that can be used to train our gesture generator in a VAE setting. To our knowledge, we are the first to discuss this in the context of gesture recognition.

1.3 Organization

This dissertation is organized as follows. Chapter 2 discusses the preliminaries and presents the relevant background information upon which our research is based on, along with the mathematical notation used throughout this dissertation. Chapter 3 examines the existing work in the literature that are the most relevant to our research. Chapters 4 and 5 examine our proposed network models for segmented and unsegmented gesture recognition tasks. Chapter 6 outlines our proposed method for generating synthetic gestures using recurrent networks. Chapter 7 summarizes our work and concludes this dissertation by providing guidance on future research directions.

CHAPTER 2: BACKGROUND AND NOTATIONS

In this chapter, we focus on the background information that is relevant to the topics of gesture recognition and generation and provide some necessary information pertinent to our work. The goal is to provide readers with an overview of the topics most relevant to our research. We also introduce the mathematical notation that is used throughout this dissertation.

2.1 Definitions and Notations

We define *gestures* as temporal sequences of actions or movements, produced by a user in order to convey an intent to perform some action through a user interface (UI). These sequences can be collected from a variety of sources, and can be represented in different forms: 2D coordinates of pen, mice or touch devices, 3D skeletal features collected by Kinect or Leap Motion devices, sonar or Doppler soundwaves, audio or video *etc*. For the purposes of this research the source of a gesture or its low-level representation in terms of features is unimportant. Thus, our definition of gestures is general and not specific to one modality or representation.

The representation of a gesture datum can be done in a variety of ways. In this dissertation we choose to represent them as a temporal sequence of the underlying feature data (*e.g*. 3D joint positions, accelerometer or velocity measurements, 2D Cartesian coordinates of pen/touch interactions, *etc*.). At time step t, the gesture data is the column vector $x_t \in \mathbb{R}^N$, where N is the dimensionality of the feature vector. Thus, the entire temporal sequence of a single gesture sample is the matrix $\mathbf{x} \in \mathbb{R}^{N \times L}$, where L is the length of the sequence in time steps. We denote the vector trajectory path of a gesture $\mathbf{x}$ with $\vec{\mathbf{x}} = \{\overrightarrow{(x_i - x_{i-1})}\ ,\ \ \forall x_{i>0} \in \mathbf{x}\}$. Lastly, we use $|A|$ to denote the cardinality of a point set A, thus in this work $|\mathbf{x}| = L$ and $|\vec{\mathbf{x}}| = L - 1$.

The dimensionality N depends on the device that generated the data and also how one chooses to represent the data. In this sense, we design our gesture recognition or generation methods to be agnostic to the input representation. For instance, consider a gesture sample collected from a Kinect device. This gesture sample might have the 3D position of 21 joints of a human actor's skeleton performing an action in L time steps. One can take N to be 3×21=63 dimensional and represent this sample as $\mathbf{x} \in \mathbb{R}^{63\times L}$. Now consider a variation of this gesture sample that involves two human actors. In this case, one can take N to be $2\times3\times21$=126 dimensional (the sample as $\mathbf{x} \in \mathbb{R}^{126\times L}$). Alternatively, one may choose to interleave the human skeletons temporally[1]. In this case, the dimensionality of N would still be 63, however, the gesture sample itself would have double the number of time steps, making the sample $\mathbf{x} \in \mathbb{R}^{63\times 2L}$.

Note that various input sequences could have different numbers of time steps. We treat sequences of different lengths in two different manners, depending on the application or the method. For recognition tasks, which mostly work on batches of data due to training phase requirements, we represent the i^{th} batch of the gesture data as the tensor $\mathbf{X}_i \in \mathbb{R}^{B\times N\times \widetilde{L}}$, where B is the size of the batch and $\widetilde{L}$ is the length of the longest sequence in the i^{th} batch. Sequences that are shorter than $\widetilde{L}$ are zero-padded. As such, for the task of recognition we use all available data from every gesture. For gesture generation tasks, however, we define $\widetilde{L}$ to be the *target length* of all gestures, and we uniformly resample equidistant points along each gesture path, making all available data to be of the same length $\widetilde{L}$.

2.1.1 Gesture Recognition

Given a dataset of gestures, we intend to define the goal for the task of gesture recognition. Formally, let us assume a dataset of application-specific gestures $\mathbb{D}$ is given, and it consists of n

[1]We choose to use this representation when working with multi-actor gestures in Chapter 4.

gesture samples $\mathbf{x}$, where $\mathbb{D} = \{\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_n\}$ and $\mathbf{x} = \{x_0, x_1, ..., x_{L_\mathbf{x}-1}\}$. Also each sample $\mathbf{x}$ is assigned a label, and there are m different label classes comprising the label set Y where $Y = \{y_0, y_1, ..., y_m\}$. The goal of gesture recognition is determine the correct class for a given gesture, *i.e.* map an unlabeled $\mathbf{x}_i$ to a label y_j. While this definition is very general, in this dissertation we are mostly interested in two types of recognition: *segmented gesture recognition* and *unsegmented gesture recognition*. Note that in this dissertation, we use the terms *unsegmented gesture recognition*, *continuous gesture recognition* and *online gesture recognition* interchangeably.

In segmented gesture recognition, each sample $\mathbf{x}$ has predefined boundaries and some information about the beginning and the end of the gesture is present. In such cases, gesture recognition involves assigning a class label to $\mathbf{x}$. More formally, given an arbitrary sample $\mathbf{x}$ from class y, the goal of segmented gesture recognition is to predict the class label $\hat{y} \in Y$, such that $\hat{y} = y$.

In unsegmented gesture recognition, the boundaries of $\mathbf{x}$ are not demarcated. In fact, in such scenarios we are interested in analyzing a sequence of data $\widetilde{\mathbf{x}}$ and determining **(1)** whether zero or more gestures are present (gesture spotting), **(2)** what the class of each gesture is (gesture recognition), and **(3)** where the start/end boundaries of the said gestures are (gesture segmentation). Thus, the recognition task differs depending on which of these questions are to be answered. In this dissertation, we refer to the collective set of these three tasks as unsegmented (or continuous) gesture recognition, which is more challenging than each individual task [6]. Formally, given an arbitrary sequence of data $\widetilde{\mathbf{x}}$ we are interested in extracting or demarcating subsequences $\mathbf{x}_i$, where each $\mathbf{x}_i$ specifies a defined gesture of class y_i. Additionally, we seek to determine the boundaries of $\mathbf{x}_i$ within $\widetilde{\mathbf{x}}$.

2.1.2 *Gesture Generation*

Gesture generation can simply be defined as producing synthetic samples over a dataset of gestures $\mathbb{D}$ such that the synthetic samples mirror data-specific properties of the samples in $\mathbb{D}$, as if these samples were seemingly sampled from $\mathbb{D}$. Formally, let us assume that $\mathbb{D}$ has the distribution $p_{\mathbb{D}}$, meaning $\mathbf{x} \in \mathbb{D} \implies \mathbf{x} \sim p_{\mathbb{D}}$. The goal of gesture generation is to produce samples $\mathbf{x}'$ where $\mathbf{x}' \notin \mathbb{D}$ but $\mathbf{x}' \sim p_{\mathbb{D}}$. This is equivalent to saying we want $p_G \simeq p_{\mathbb{D}}$, where p_G denotes the distribution of the generated samples.

Note that the definition of *generation* in this dissertation deliberately excludes common data *augmentation* methods. For instance, one could define a generator function G where $G(\mathbf{x}) = \text{reverse}(\mathbf{x})$. Although the temporal reverse of $\mathbf{x}$ could be a seemingly valid sequence, G is not considered a valid generator in our discussion because depending on $\mathbb{D}$ it is possible that $G(\mathbf{x}) \nsim p_{\mathbb{D}}$.

2.2 Neural Networks and Deep Learning

In recent years, neural networks and deep learning have received a staggering amount of attention due to their power and unreasonable efficacy in solving various computational problems. In short, neural networks which are inspired by our brains, are a set of connected nodes organized in a typically acyclic graph structure such as the one shown in Figure 2.2. These networks compute a non-linear function $\hat{y} = F(x; \theta)$, where x is the input to the network (red nodes), θ are the trainable *weights*, and $\hat{y}$ is the output (green nodes). Each node computes a non-linear combination of its inputs (the arrows), and passes the results to all the nodes that are connected to it. The term *deep* neural networks refers to networks that have more than a few layers of hidden units (blue nodes). Note that the hidden units can consist of arbitrary non-linear functions. These functions are domain specific. Traditionally, logistic regression units were used in neural networks. More

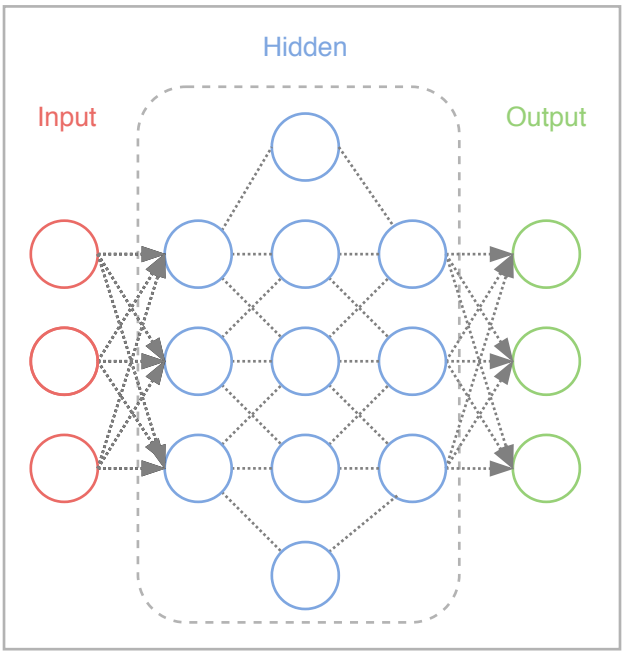


Figure 2.1: A graphical representation of a neural network

recently, convolutional units are extensively used for image perception tasks, whereas recurrent units are used for sequence processing. These units have given rise to two of the most popular types of neural networks, namely convolutional neural networks (CNN) and recurrent neural networks (RNNs – discussed shortly).

The non-linear computation carried out by each node in a neural network is governed by the trainable parameters θ. Consider a logistic regression unit: given the input $\mathbf{x}$, these units compute $\hat{y} = \sigma(w^\intercal x + b)$, where σ is the sigmoid function and w and b are the weight and bias values that comprise the trainable parameters θ of a logistic regression unit. These parameters are established during the *training* process. The training process involves determining the parameters θ such that given the input x, the *desired* output y is produced. This process is often guided by an *objective function* (the *loss* measure $\mathcal{L}$) which typically defines the relationship between the network's current output $\hat{y}$ and the desired output y, given an input x. Based on this, we can formulate the

following optimization problem to establish suitable values for the parameters θ:

$$\underset{\theta}{\operatorname{argmin}} \mathscr{L}\Big(\hat{y}, y\Big) = \underset{\theta}{\operatorname{argmin}} \mathscr{L}\Big(F\big(x;\theta\big), y\Big) \tag{2.1}$$

If F is differentiable, *i.e.* the derivative $\nabla_x F(x;\theta)$ is defined for all valid inputs x, one could solve Equation 2.1 using analytical or numerical methods. Due to the highly non-linear nature of neural networks, as well as the complexity of the cost landscape defined by most loss functions $\mathscr{L}$, numerical methods such as Stochastic Gradient Descent (SGD) or similar (such as Adam [7]) are popular choices of optimizers.

2.3 Recurrent Neural Networks

Neural networks as defined above, are typically used for tasks that do not include temporal information, as they are stateless. In contrast, *recurrent* neural networks (RNN) are designed to work with temporal data. In addition to raw feature values, these networks take as input their own output from the past, enabling them to preserve some representation of a state. Such networks are typically used for tasks such as natural language understanding, time series analysis and prediction as well as video analysis. In this dissertation, we use these networks for gesture recognition and synthesis.

2.3.1 Recurrent Units

The building blocks of RNNs are the *recurrent units*. At each timestep t, a recurrent unit $\mathcal{R}$ takes as input the feature vector x_t and a hidden state vector $h_{(t-1)}$, and produces an output vector which

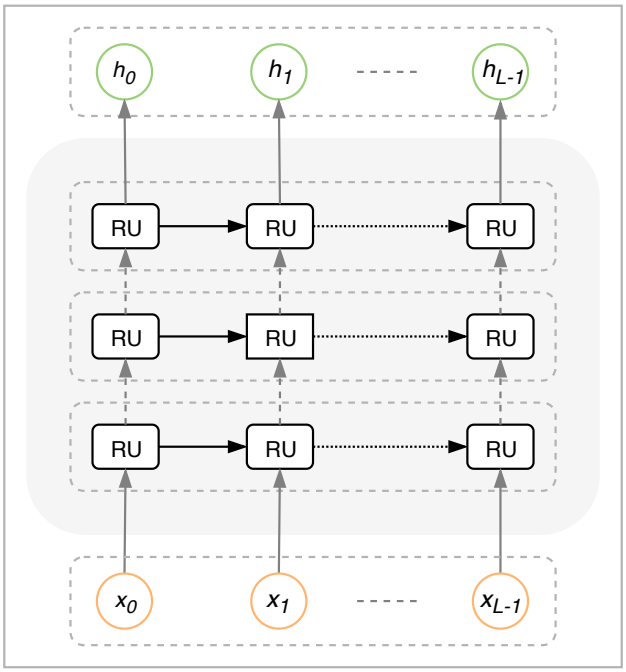


Figure 2.2: A graphical representation of a recurrent neural network. *RU* denotes a recurrent unit.

is a new hidden state h_t. This transition can be shown as:

$$h_t = \mathcal{R}(x_t, h_{(t-1)}; \theta) \tag{2.2}$$

where θ are the trainable parameters of $\mathcal{R}$. The hidden state h_t is subsequently used for performing the computations on $x_{(t+1)}$, transferring the state along the timesteps. The initial hidden state vector h_0 is often set to a vector of zeros. A RNN is typically built by stacking recurrent units in layers such that the output of one unit becomes the input to another unit in the next layer as shown in Figure 2.3.1. Note that the general representation presented above assumes a *unidirectional* RNN: a network that processes a given sequence $\mathbf{x}$ from $t = 0$ all the way to $t = L - 1$. In the literature, the *bidirectional* RNNs, which process a sequence both forwards and in reverse also exist, but they are beyond the scope of this dissertation.

Although different types of recurrent units exist in the literature, in this dissertation we focus on two of the most common ones, namely Long Short-Term Memory (LSTM) [8] and Gated Recurrent Units (GRU) [9]. Both of these units are *gated*, meaning these units internally contain decision logic as to when an internal state update should occur. This decision is guided by a transformation of the input feature vector x_t (via the learned parameters), as well as the values in the hidden state vector $h_{(t-1)}$.

Originally proposed for neural machine translation [9], GRUs are one of the most popular recurrent units suitable for various other tasks such as music generation and speech signal modeling [10]. In fact, as Chung *et al*. note, the power of GRUs is comparable to that of LSTMs. While less computationally intensive, these units can achieve better performance than LSTMs on some small datasets. In this dissertation we use the notation $\mathbf{\Gamma}$ to denote a GRU.

Various implementations and state transitions exist in the literature. In this dissertation we use the following transition equations, which are also what the PyTorch framework [11] uses.

$$
\begin{aligned}
h_t &= \mathbf{\Gamma}\big(x_t, h_{(t-1)}\big) \\
r_t &= \sigma\ \Big(\big(W_x^r\ x_t + b_x^r\big)\ +\ \big(W_h^r\ h_{(t-1)} + b_h^r\big)\Big) \\
u_t &= \sigma\ \Big(\big(W_x^u\ x_t + b_x^u\big)\ +\ \big(W_h^u\ h_{(t-1)} + b_h^u\big)\Big) \\
c_t &= \tanh\Big(\big(W_x^c\ x_t + b_x^c\big)\ +\ r_t\big(W_h^c\ h_{(t-1)} + b_h^c\big)\Big) \\
h_t &= u_t \circ h_{(t-1)}\ +\ \Big(1 - u_t\Big) \circ c_t
\end{aligned}
\tag{2.3}
$$

In the equation above, σ is the sigmoid function, $\circ$ denotes the Hadamard product, r_t, u_t and c_t are reset, update and candidate gates respectively and W_p^q and b_p^q are the trainable weights and biases.

What these equations intuitively represent is that a candidate output value c_t is computed from the current input x_t and the previous output $h_{(t-1)}$. This candidate value is computed based on one non-linear computation from the reset gate r_t and one linear computation from the candidate's trainable weights W^c, b^c. Note that the reset gate r_t itself depends on two other sets of trainable weights, namely W_x^c, b_x^c and W_h^c, b_h^c. Lastly, the output value h_t is gated (governed) by the update gate u_t. Namely, h_t is set to be equal to the last output $h_{(t-1)}$ with a probability of u_t, and is set to be the candidate value c_t, which is a newly computed transformation of the inputs, with a probability $(1 - u_t)$. As we show later in Chapter 4, the decision mechanism of a GRU and the candidate value c_t can be used to govern whether a particular feature vector should undergo additional transformation and feature extraction.

Similar to GRUs, the LSTM units have been shown to demonstrate great performance in a variety of tasks [10]. The power of these units comes at the cost of extra processing power requirements, and they are more prone to overfitting. The gating mechanism of the LSTMs, which is similar to GRUs but contains additional trainable weights and decision logic, is out of the scope of our discussion. Consequently, in this dissertation we prefer GRUs over LSTMs due to their power, simplicity, and the fact that they do not overfit as much as LSTMs.

2.3.2 Attention Model

When working with sequential data, the sub-parts of a temporal sequence may not all be equally important: some subsequences may be more pertinent to the task at hand than others. Thus, it is often beneficial to learn a representation that can identify these important subsequences and leverage them to tackle the subject matter. This is the key intuition behind the attention model [12, 13]. Even though the attention model was originally proposed for sequence to sequence models and neural machine translation, it has been adapted to various tasks involving the processing of

sequential data [14]. Unsurprisingly, it has also been adapted to the task of gesture and action recognition [15–20].

The attention model intuitively works as follows. During the process of feeding a sequence into the RNN, when the vector x_t at timestep t is being processed, the attention model examines the generated outputs in the vicinity of t, and determines which $h_i \in [h_{t-\delta}, h_{t+\delta}]$[2] would aid in the processing of x_t when producing the output h_t. Note that these models typically affect the output h_t of a RNN.

The attention model learns how to determine the importance of each sub-sequence during training. A formal definition follows. Let t denote the timestep that the RNN is currently processing, and t' denote what has already been processed so far.

At timestep t, the model computes a set of *attention weights* $\alpha_{tt'}$, using which it can generate a dynamic representation of the relevance of the computations at timestep t' when processing the current timestep t. In simple terms, $\alpha_{tt'}$ is the amount of attention that the output at t should pay to the output at t'.

The weights $\alpha_{tt'}$ are used to compute a context vector c_t, the output of the attention model for timestep t. Given $a_{t'}$ the extracted features at a timestep t' and h_{t-1} the latest hidden state the RNN (the latest output that is generated so far), the following equations detail how the attention model computes the context vector c_t:

[2]Note that $t + \delta$ is applicable to bidirectional RNNs, which we do not use in this dissertation.

$$e_{tt'} = \mathbf{F}_{\text{attn}}\Big(a_{t'}, h_{(t-1)}; \theta\Big) \tag{2.4}$$

$$\alpha_{tt'} = \text{softmax}\big(e_{tt'}\big) = \frac{\exp\big(e_{tt'}\big)}{\sum_{k=0}^{L-1} \exp\big(e_{tk}\big)} \tag{2.5}$$

$$c_t = \sum_{t'} \alpha_{tt'} \cdot a_{t'} \tag{2.6}$$

where $\mathbf{F}_{\text{attn}}$ is a small neural network with trainable parameters θ. Training the attention model involves optimizing for the parameters θ such that the model can better determine the most relevant parts of a sequence.

2.4 Generative Modeling with Deep Networks

In this section we turn our focus to the application of deep networks for generative modeling. Such models, which typically fall under the category of unsupervised learning, aim to learn and model a representation of the data so that new data points could be sampled from this learned representation. In this work, we focus on two of the most relevant generative models, namely *generative adversarial networks* and *variational autoencoders*.

2.4.1 Generative Adversarial Networks

It is difficult to overstate the machine learning community's interest in generative adversarial networks (GAN). Introduced by Goodfellow *et al*. [21] these generative class of deep networks are proficient in generating fake data by learning what the real data look like. These unsupervised models have been extensively used for image generation tasks (such as fake human faces, *etc*.) [22].

GANs typically consist of two deep networks: a *generator* and a *discriminator*. The goal of the discriminator is to determine whether a given sample is real or fake, and the goal of the generator is to fool the discriminator into thinking that the fake samples are actually real. Both networks are trained at the same time during adversarial training: the goal of each network contradicts the other's. When training ends, the generator model is kept and used, while the discriminator network is often discarded.

Note that typically the generator takes as input a vector of random noise (referred to as the *latent* vector) and produces a synthetic sample as the output. As such, the generators can be thought of as functions that take as input a probability distribution and map it to the underlying distribution of the dataset. Moreover, different variations of GANs exist in the literature. The most relevant to our research are conditional GANs [23]. These models parameterize the generator network on various conditions, most commonly on class labels. In other words, in addition to the latent vector, they take a condition and generate a sample that satisfies the condition (*e.g.* conditioning a face generation network can be done by mandating that the generated face should be of a male who is wearing sunglasses). Conditions are often incorporated in the latent vector by appending a representation of the condition to the vector.

Let us now formalize GANs as pertinent to the task of gesture generation. Given a dataset $\mathbb{D}$ of gesture samples, our goal is to learn the distribution of the samples through training a recurrent neural network which we refer to as the *generator* (G) hereto. Once trained, G can generate new samples, and the newly sampled instance should be a good representative of the samples in $\mathbb{D}$. More formally, G can generate synthetic temporal sequences of length L, and is a function that maps a vector of random noise $\mathbf{z} = \{z_0, z_1, ..., z_{L-1}\}$ (the latent variable) to a synthetic sample $\mathbf{x}' = \{x_0, x_1, ..., x_{L-1}\}$. We show this mapping via $\mathbf{x}' = G(\mathbf{z}; \theta_G)$ where θ_G are the trainable parameters that can represent an estimate of the distribution of $\mathbb{D}$. For simplicity, we refer to $G(\mathbf{z}; \theta_G)$ as $G_\theta(\mathbf{z})$ henceforth. Since $\mathbb{D}$ can contain samples of different classes, the latent variable

$\mathbf{z}$ is assumed to be conditioned on the class that we intend to generate the synthetic samples for. We follow the convention in [24] for conditioning: the intended condition c is appended to the latent vector $\mathbf{z}$ at each timestep such that $\mathbf{z}' = \{\left[\ z_i\ ;\ c\ \right]\ \ \forall z_i \in \mathbf{z}\}$. Henceforth, when referring to the latent vector $\mathbf{z}$, we are actually referring to the conditioned vector $\mathbf{z}'$ (if applicable). Also, if the condition is a class label, we use the "one-hot" representation of the class label as the condition c in this dissertation.

The typical practice in GAN research is to learn θ_G through adversarial training, where θ_G is estimated based on how well a discriminator network D can distinguish between real and fake samples. Given the output G, the discriminator D computes:

$$\tilde{y} = D\big(G_\theta(\mathbf{z}); \theta_D\big) \tag{2.7}$$

where $\tilde{y} \in \big\{\text{real}, \text{fake}\big\}$ and θ_D are the trainable parameters. For simplicity, we refer to $D(\mathbf{x}; \theta_D)$ as $D_\theta(\mathbf{x})$ henceforth. The *goodness of fit* measure here is the adversarial loss that one tries to minimize. Because our underlying data are time series, the natural choice for G are RNNs.

2.4.1.1 Training

The original GAN model formulates the training process as described previously: G should be optimized such that it fools D, whereas D is optimized to better distinguish between real and fake samples. Unfortunately, this formulation of GANs is susceptible to a few pitfalls. Most importantly, the training of GANs with this formulation is often unstable: one network could easily overpower the other, resulting in non-convergence. Also, care must be taken when selecting the neural architecture for each network. Also, one of the most common problems is *mode collapse* which refers to the lack of variety in the generated samples. This often happens when the generator

constantly tries to fool the discriminator with a few limited samples.

Many approaches have been proposed in the literature [25] to overcome these difficulties, including numerous loss function formulations. Although a recent paper has experimentally showed that there may not be significant differences among different GAN training losses [26], we have experienced great success in our work with two particular losses: Wasserstein loss [27] and its improved version with gradient penalties [28].

2.4.1.2 Wasserstein GANs

Rather than formulating GAN training as a zero-sum game, Arjovsky *et al*. [27] formulate the problem as the direct estimation of the distribution of dataset $\mathbb{D}$ with the generator G. They do this by defining a loss measure, namely the *Wasserstein loss* that gauges the similarity of the distribution of the generated samples $\mathbf{x}'$ with the real ones $\mathbf{x}$ by means of a *critic* network D. Arjovsky *et al*. experimentally verified that Wasserstein GANs were less prone to issues such as mode collapse, and were generally easier to train across a variety of tasks. Formally, the Wasserstein GAN minimizes the loss values $\mathscr{L}_D$ and $\mathscr{L}_G$, respectively for D and G as follows:

$$\begin{aligned} \mathscr{L}_D &= D_\theta\big(\mathbf{x}'\big) - D_\theta\big(\mathbf{x}\big) \\ \mathscr{L}_G &= -D_\theta\Big(G_\theta\big(\mathbf{z}\big)\Big) \end{aligned} \tag{2.8}$$

Arjovsky *et al*. [27] show that for networks D that are 1-Lipschitz functions, minimizing $\mathscr{L}_D$ is equivalent to minimizing the Wasserstein distance between $p_{\mathbb{D}}$ and p_G. Enforcing Lipschitz constraints in D is achieved by clipping the weights θ of D, which in Arjovsky's *et al*. words is

a "terrible way to enforce a Lipschitz constraint"! Later, Gulrajani *et al*. [28] introduced the improved Wasserstein GANs to better address the requirement for Lipschitz continuity in D. Namely, they introduced the *gradient penalty* measure as a means of enforcing these constraints in D, and reformulated the losses above as follows:

$$\begin{aligned} \hat{\mathbf{x}} &= \epsilon \mathbf{x} + (1-\epsilon)\mathbf{x}' \\ \mathscr{L}_D &= D_\theta\big(\mathbf{x}'\big) - D_\theta\big(\mathbf{x}\big) + \lambda\Big(\Big\|\nabla_{\hat{\mathbf{x}}} D_\theta(\hat{\mathbf{x}})\Big\|_2 - 1\Big)^2 \\ \mathscr{L}_G &= -D_\theta\Big(G_\theta\big(\mathbf{z}\big)\Big) \end{aligned} \tag{2.9}$$

where $\epsilon \in \mathcal{U}(0,1)$. As detailed in [28], $\mathscr{L}_D$ measures how far the distribution of the synthetic samples is from the real ones, regularized by a gradient penalty term, the goal of which is to ensure a gradient norm of 1 on D, which in turn enforces the Lipschitz constraints. The gradient penalty term is controlled by the coefficient λ, which is a hyperparameter for the optimization problem.

2.4.1.3 Application in Domain Adaptation

An interesting application of GANs is towards *domain adaptation*. In machine learning, the need for domain adaptation arises when there is a distribution mismatch between the data on which a model is trained on (source domain), and the data on which the model is applied to (target domain). In such cases, the effect of *domain shift* causes a model which performs well in the source domain to perform poorly in the target domain.

Recently, GANs have been used towards domain adaptation [29–31]. This is done through training a generative model which maps the samples of the source domain to the target domain, thereby

reducing the effect of domain shift. Among these methods, CycleGAN [31] has shown great performance in various domain adaptation tasks [30, 32, 33].

In essence, CycleGAN performs an unpaired mapping operation from a source domain $\mathcal{X}$ to a target domain $\mathcal{Y}$ via two mapping functions $G : \mathcal{X} \rightarrow \mathcal{Y}$ and $F : \mathcal{Y} \rightarrow \mathcal{X}$. Internally, both G and F are the generators of two GAN models which are trained using their associated discriminators $D_{\mathcal{Y}}$ and $D_{\mathcal{X}}$ respectively. These discriminators encourage the indistinguishable translation of samples into their respective domains. Therefore, training CycleGAN involves training two GAN models at the same time. Additionally, CycleGAN introduces *cycle consistency losses*. These losses enforce the intuition that for a sample $\mathbf{x} \in \mathcal{X}$ the mapping $F(G(\mathbf{x}))$ should yield a sample which closely resembles $\mathbf{x}$. Cycle consistency is enforced for both mappings $\mathcal{X} \rightarrow \mathcal{Y} \rightarrow \mathcal{X}$ and $\mathcal{Y} \rightarrow \mathcal{X} \rightarrow \mathcal{Y}$.

Assuming that $\text{GAN}_{\mathcal{Y}} : \mathcal{X} \rightarrow \mathcal{Y}$ and $\text{GAN}_{\mathcal{X}} : \mathcal{Y} \rightarrow \mathcal{X}$, the following describes the loss function for training a CycleGAN model:

$$\mathscr{L}_{\text{CycleGAN}} = \mathscr{L}_{\text{GAN}_{\mathcal{Y}}} + \mathscr{L}_{\text{GAN}_{\mathcal{X}}} + \lambda \mathscr{L}_{\text{cyc}}\big(G, F\big) \tag{2.10}$$

where λ is the regularizer for the cycle consistency loss. Note that the type and the training process of $\text{GAN}_{\mathcal{X}}$ and $\text{GAN}_{\mathcal{Y}}$ is unimportant. In other words, one could select any GAN model (WGAN, WGAN-GP *etc*.) as the mapping function for training a CycleGAN model. Once a CycleGAN model is trained, one could use the trained generators (mappers) G and F to map arbitrary samples from $\mathcal{X}$ to $\mathcal{Y}$ or vice-versa.

Although minimizing Equation 2.10 typically yields well-performing mappers, Zhu *et al*. [31] found that the use of an additional loss term that enforces *identity mapping* is often beneficial for some problem domains. In simple terms, the identity loss term regularizes the generator to be almost an identity mapping when samples $\mathbf{x} \in \mathcal{Y}$ are mapped to the same domain $\mathcal{Y}$.

2.4.2 Variational Autoencoders

We briefly discuss autoencoders [34, 35] upon which variational autoencoders [36] are built. Autoencoders, which consist of encoder and decoder networks, are unsupervised learning models that are typically used for dimensionality reduction or noise removal from data. The encoder network in these models takes the input data and transforms them into a vector in a lower dimensional space (encoding). The decoder network takes the encoded data and aims to reconstruct the original data. The training process involves optimizing the parameters of the encoder and decoder networks to minimize the information lost after the input data is reconstructed. Once training concludes, the encoder network can be used to perform dimensionality reduction or de-noising of the input.

Variational autoencoders (VAE) [36] are a class of autoencoders designed for generative modeling. Similar to autoencoders, VAEs consist of encoder and decoder networks. However, the goal of VAEs is to model the complex distribution of the input data by learning a latent representation thereof. As such, the encoder network maps the input data $\mathbf{x}$ to a probability distribution (latent space) while the decoder network aims to reconstruct the original data from a vector $\mathbf{z}$ in that latent space. The loss function for VAEs consists of reconstruction as well as regularization terms. The reconstruction term ensures that the reconstructed data closely resembles the input data. The regularization term ensures that the learned distribution of the latent space is as close to the standard normal distribution $\mathcal{N}(0, 1)$ as possible.

Assuming that ϕ and θ denote the trainable parameters of the encoder and decoder respectively, the following is the loss function that is minimized to train VAEs [37]:

$$\mathscr{L}(\theta, \phi; \mathbf{x}, \mathbf{z}) \;=\; \underbrace{-\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\big[\log p_\theta(\mathbf{x}|\mathbf{z})\big]}_{\text{reconstruction}} \;+\; \underbrace{D_{KL}\big(q_\phi(\mathbf{z}|\mathbf{x}) \;||\; p(\mathbf{z})\big)}_{\text{regularization}} \tag{2.11}$$

where $D_{KL}(\ ||\ )$ is the Kullback-Leibler (KL) divergence between two probability distributions (a measure of the *difference* between the two). To put Equation 2.11 simply, the reconstruction term aims to maximize the log-likelihood of observing the input sample $\mathbf{x}$ if the decoder is fed with the latent vector $\mathbf{z}$. As for the regularization term, recall that it aims to ensure the latent space follows the standard normal distribution ($p(\mathbf{z}) = \mathcal{N}(0, 1)$). Thus, the regularization term ensures that when the encoder encodes $\mathbf{x}$ into $\mathbf{z}$, the obtained latent space follows the standard normal distribution. Note that by minimizing $\mathscr{L}$, we essentially minimize a negative log-likelihood term, and minimize the KL-divergence between the distribution of the latent space and the standard normal distribution.

Notice that with Equation 2.11, encoder and decoder networks do not distinguish between different inputs $\mathbf{x}$. This implies that we cannot target specific samples $\mathbf{x}$. For instance, if the VAE model is trained on face images, one cannot specifically ask for the generation of specific types of faces (male faces, female faces, *etc*.). To overcome this limitation, Sohn *et al*. [38] proposed conditional VAEs in which a conditioning criterion is applied to the input data $\mathbf{x}$ as well as the latent vector $\mathbf{z}$ similar to conditional GANs as described in Section 2.4.1.

2.4.3 Evaluation of Generative Models

Since generative models are capable of synthesizing fake data, an important step in evaluating them is to determine the quality of the generated data. At the time of this writing, there is no consensus on measures for the evaluation of generative models. As such, researchers rely on a combination of different measures to determine the quality of the generated outputs. Popular automatic measures include Inception Score (IS) [25] or Frechét Inception Distance (FID) [39].

IS describes the quality of the generated samples by how well the Inception v3 network [40] classifies each one. This measure, however, does not compare the generated samples against the real ones, which is what FID attempts to address. Similar to IS, FID also uses the Inception v3 model.

However, instead of a measure based on the classification confidence, it relies on the feature values computed at the last pooling layer of the Inception model. Specifically, FID computes these activation values for a collection of real and fake samples. It then computes two multivariate Gaussians, one for real feature values and one for the fake ones. Lastly, it compares the two Gaussian models using the Frechét distance, producing a measure describing how the fake samples compare against the real ones. Note that both IS and FID were introduced for image data which limits their application for generative models that produce other types of data such as gestures. Thus, neither of these measures is usable for the current work.

One popular approach of gauging the quality of generative models is conducting large-scale user-studies with human participants. Such evaluations can cover a broader range of generated data types, making them more appropriate for the current work. Although there are various ways to design and run such studies, Zhou *et al*. recently introduced the HYPE_{∞} benchmark [41] which defines a gold standard benchmark for evaluating generative realism on crowd-sourcing platforms. This measure defines the quality of generative models' outputs as the percentage of samples that were judged incorrectly by human participants. For instance, a HYPE_{∞} value of 20% means that the understudied generator can fool human participants 20% of the time. HYPE_{∞} values of above 50% indicate *hyper-realism*: generated samples look more realistic to humans than real samples.

To compare different generative models, HYPE_{∞} defines a between-subject design with 30 participants: each participant only sees the results from one of the models. For a given generative model, every participant is shown a total of 100 samples comprised of 50 fake and 50 real samples. Given each sample, participants are asked to indicate whether they think that sample is real or computer-generated without any time limits. Afterwards, the HYPE_{∞} value is computed for each participant and is averaged across 30 participants. Zhou *et al*. [41] showed that this protocol ensures repeatability and maintains the separability between different generative models and can be used as a reliable measure of the generative model's quality.

2.5 Dynamic Time Warping

Dynamic time warping (DTW) is a dynamic programming algorithm that was originally proposed for speech recognition [42], yet has found application in many fields involving data time series analysis [43]. It is a dissimilarity measure of two time series that can be used to find the best alignment of the two. As such, DTW was recently shown to be an effective nearest-neighbor matching measure for gesture recognition [44].

DTW works as follows. Given two time series $\mathbf{X} = \{x_1, x_2, ..., x_n\}$ and $\mathbf{Y} = \{y_1, y_2, ..., y_m\}$, a cost matrix Δ of size $n \times m$ is built[3]. Each element Δ_{ij} is the matching cost of x_i to y_j, computed via the following equation, which can be implemented via dynamic programming:

$$\Delta_{ij} = d\big(x_i, y_j\big) + \min\Big\{ \underbrace{\Delta_{i-1,j}}_{\text{insertion}} \;,\; \underbrace{\Delta_{i,j-1}}_{\text{deletion}} \;,\; \underbrace{\Delta_{i-1,j-1}}_{\text{matching}} \Big\} \tag{2.12}$$

where $d\big(x_i, y_j\big)$ is a cost (distance) function that is problem-specific. Although Euclidean distance (ED) between x_i and y_j is widely used, Taranta *et al.* [44, 45] demonstrated the superiority of using the cosine similarity measure (COS) for gesture recognition problems. The notations in Equation 2.12 denote what each term of the cost matrix Δ intuitively means. Once Δ is fully computed, the value Δ_{nm} is the dissimilarity measure of $\mathbf{X}$ and $\mathbf{Y}$, and the path through the matrix that yields Δ_{nm} is the optimal alignment between the two time series.

[3]For simplicity, in this work we always assume that $\mathbf{X}$ and $\mathbf{Y}$ are of the same length.

2.5.1 *Differentiable (Soft) Dynamic Time Warping*

A closer look at Equation 2.12 reveals that DTW is not differentiable, primarily because of the lack of smoothness in the recurrence, caused by the computation of a minimum value. As such, at the first glance one might assume that DTW cannot be used in gradient-based optimization algorithms.

To address this shortcoming, recently Cuturi and Blondel [46] proposed a differentiable formulation of DTW called *soft dynamic time warping* (sDTW). The soft formulation of DTW replaces the minimum computation with a differentiable soft minimum computation that is controlled by a smoothing parameter γ:

$$\min^{\gamma}\left\{a_1, a_2, ..., a_n\right\} = \begin{cases} \min(a_i), & \gamma = 0 \\ -\gamma \log \sum_{i=1}^{n} e^{-a_i/\gamma}, & \gamma > 0 \end{cases} \tag{2.13}$$

From Equation 2.13, it is clear that using $\gamma = 0$ would result in the original minimum value computation, yielding the DTW formulation as presented in Equation 2.12. Note that larger values for γ cause more smoothness in the shape of the resulting function. For values $\gamma > 0$, Cuturi and Blondel show that the value Δ_{nm} would actually be the expected value of the dissimilarity between $\mathbf{X}$ and $\mathbf{Y}$, over every possible alignment between them weighted by their probability under the Gibbs distribution [46].

CHAPTER 3: RELATED WORK

In this chapter, we provide an overview of the related work in the literature. We examine the prior research in gesture recognition and review the state of research in gesture generation. We also highlight how this research differs from all other prior work.

3.1 Gesture Recognition

Gesture recognition has been a long-standing problem. The related field of sketch recognition dates all the way back to the work of Sutherland on Sketchpad [47]. Arguably, the work of Rubine [48] was the first rigorous attempt at recognizing gestures specified via examples. Rubine showed that by extracting discriminant features from gesture samples, a simple linear recognizer could achieve respectable recognition accuracies. Geometric features such as initial angle, length of the gesture, size of the bounding box diagonal, among others, helped Rubine's recognizer in distinguishing between different 2D gestures.

Today, as with most other problems involving recognition by example, the problem of gesture recognition is approached with solutions that come from two general categories: recognition with hand-crafted features, or recognition through automatic extraction of features by learning good features from the data itself. We refer to the latter

3.1.1 Recognition with hand-crafted features

Methods that attempt recognition via hand-crafted features have demonstrated great success over the years [49–54] and thus fall into the classical categories of recognition. Methods in this category

often design domain-specific discriminant features that are extracted from a set of training samples. These features are then fed to a model that can find a decision boundary between target and non-target classes, and thus recognize gesture samples that belong to a particular class.

As Cheema *et al*. [55] showed, these methods can achieve excellent recognition results. They compared the performance of five algorithms (AdaBoost, SVM, Bayes, decision trees, and the linear classifier) on Wii controller gestures and concluded that, in some cases, the seemingly simple linear classifier can recognize a set of 25 gestures with 99% accuracy.

An overview of successful methods in this category follows. Weng *et al*. [56] leveraged the spatio-temporal relations in action sequences with naïve-Bayes nearest-neighbor classifiers [57] to recognize actions. Xia *et al*. [3] used hidden Markov models (HMM) and the histogram of 3D joint locations to recognize gestures. Vemulapalli *et al*. [58] represented skeletal gestures as curves in a Lie group and used a combination of classifiers to recognize the gestures. Wang *et al*. [59] modeled the spatio-temporal motion properties of joints with a graph of motionlets. These graphs were then classified using SVMs to recognize actions. Similarly, De Smedt *et al*. [60] used SVMs to recognize hand gesture trajectories represented as data lying on Riemannian manifolds. Evangelidis *et al*. [61] proposed skeletal quads, a skeleton descriptor which encodes the relative position of joint quadruples which were then used for classifying actions.

3.1.2 The $-Recognizer Family

Although the recognizers described above can demonstrate great recognition performance, one prohibiting factor in their adoption is the complexity of their implementation. Imagine a mobile application developer who likes to integrate gesture-based interactions in their app, but finds the task daunting due to the requirement for hand-engineering features, or implementing a machine learning algorithm. Although off-the-shelf implementations of various machine learning algo-

rithms are easily accessible nowadays, the odds are good that a lot of effort from the developer is needed to train, evaluate and integrate the algorithm in their app.

The $-family of recognizers [62–68] and their extended family [44, 45, 69–73] were introduced to address these challenges. These recognizers attempt to provide an accessible means of incorporating gesture-based interactions in applications and prototypes. Simplicity and ease of implementation and use are the cornerstones of the design philosophy of these recognizers. Most of these recognizers can be easily understood, without the need for a lot of domain knowledge, and can be implemented in a few lines of code in a variety of programming languages. Additionally, these recognizers are suitable for *gesture customization*: the ability to allow users to define their own gestures by providing a few examples of the gestures they intend to use. This implies that the $-family of recognizers can be trained with just a few training samples.

The design philosophy of these recognizers implies that they typically do not rely on hand-engineered features. In fact, most of these recognizers are *template-based*: internally they maintain a set of gesture templates, and given a gesture sample they attempt to match it to one of those internal templates. These recognizers mostly differ in input data representation and matching criteria. The $1 recognizer [62] uses Euclidean distance to match gestures to a template, while Protractor [63] uses angular information for the matching process. Alternative dissimilarity measures such as the cosine or DTW dissimilarity have also been explored in Penny Pincher [45] and Jackknife [44]. The $N recognizer [64] and $N-Protractor [65] are extensions of $1 and Protractor, respectively, which add support for multi-stroke gestures. Similarly, the $3 recognizer [70] and Protractor 3D [71] are extensions of $1 and Protractor, respectively, which add support for 3D input data.

Although most of the recognizers in the original $-family represent their inputs as ordered sets of points, alternative representations have also been explored. The $P [66] and $P+ [67] recognizers represent their inputs as point clouds which enables better recognition performance for

both unistroke and multistroke gestures. Their extension, $Q [68], was introduced to speed up the recognition performance in the point cloud data domain. The use of direction vectors as the input data representation has also been explored. Specifically, Penny Pincher [45] and Jackknife [44] showed the effectiveness of such representation in various gesture recognition problems.

3.1.3 Deep Networks

Although most deep learning methods rely on automatic feature extraction from the input, the use of hand-crafted features to train such networks has been previously explored. Chen *et al*. [60] proposed the use of hand-crafted features along with raw skeleton data to recognize hand gestures. Similarly, Avola *et al*. [74] used a LSTM architecture in conjunction with hand-crafted angular features of hand joints to recognize hand gestures. Ke *et al*. [75] proposed the use of translation, rotation and scale-invariant features extracted from the body parts in each frame to learn a more robust representation of each gesture. This information was then transformed into image data and fed to a convolutional neural network (CNN) to perform gesture recognition. The use of convolutional operations for gesture recognition has also been explored in the work of Devineau *et al*. [76], which proposed the use of parallel convolutions to extract features from hand joints. These extracted features were then fed to a multi-layer perceptron to perform recognition.

In contrast, we aim to solely use automatically extracted features to perform gesture recognition using RNNs. We explicitly avoid using hand-crafted features because such features are inherently domain-specific which contradicts our design philosophy.

More recently, RNN-based models have been widely used for gesture recognition. Shahroudy *et al*. [77] showed the power of recurrent architectures and long-short term memory (LSTM) units [8] for large-scale gesture recognition. Zhang *et al*. [78] proposed a view-adaptive scheme to achieve view-invariant action recognition. Their model consisted of LSTM units that would learn the most

suitable transformation of samples to achieve consistent viewpoints. Liu *et al*. [79] incorporated the spatio-temporal and contextual dependencies to recognize actions from 3D skeletons. The contextual updating mechanism of their LSTM units was further controlled by a gating mechanism which improved robustness. The use of ensemble methods has also been explored in the literature. Lee *et al*. [80] used an ensemble of LSTMs to capture the short-term, medium-term and long-term dependencies between input features to recognize actions from skeletal data. Núñez *et al*. [81] used a combination of CNNs and LSTMs with a two-stage training process to classify skeleton and hand gestures. Weng *et al*. [82] proposed the use of one-dimensional convolutions to extract features from skeleton joints. Classification was done using a LSTM model.

Most such prior work relies on LSTMs internally. In contrast, we only use gated recurrent units (GRU) [9] as the building block of our recurrent network. It has been shown [10] (and we also show later), that GRUs are faster to train and sometimes obtain better results. Also, our method is designed to be general and not specific to a particular device, gesture modality or feature representation. Lastly, we leverage the attention mechanism to capture the most important parts of each input sequence.

The attention model has also been used towards gesture and action recognition. Liu *et al*. [19] proposed a global context-aware attention LSTM network for 3D action recognition. Using a global context, their method selectively focuses on the most informative joints when performing recognition. Song *et al*. [20] used the attention mechanism with LSTM units to selectively focus on discriminative skeleton joints at each gesture frame. Fan *et al*. [18] introduced a multi-view re-observation LSTM network which augments any observed action with multiple views of the same action in order to achieve view-invariant recognition. Baradel *et al*. [16] proposed a two-stream convolutional and LSTM network which used pose as well as image information to perform action recognition. They demonstrated the importance of focusing on the hand motion of the actors in the sequence to improve recognition accuracy. Later, Baradel *et al*. [17] leveraged the visual attention

model to recognize human activities purely using image data. They used GRUs as the building block of their recurrent architecture.

Contrary to some of this work, DeepGRU only requires pose and vector-based data. Our novel attention model differs from prior work in how the context vector is computed and consumed. For instance, GCA-LSTM [19] has a multi-pass attention sub-network which requires multiple initialize/refine iterations to compute attention vectors. Ours is single-pass and not iterative. Our attention model also differs from STA-LSTM [20] which has two separate temporal and spatial components, whereas ours has only one component for both domains. VA-LSTM [78] has a view-adaptation sub-network that learns transformations to consistent view-points. This imposes the assumption that input data are spatial or view-point dependent, which may prohibit applications on non-spatial data (*e.g*. acoustic gestures [83]). Our model does not make any such assumptions. As we show later, our single-pass, non-iterative, spatio-temporal combined attention, and device-agnostic architecture result in less complexity, fewer parameters, and shorter training time, while achieving state-of-the-art results, which we believe sets us apart from prior work.

3.2 Continuous Gesture Recognition and Segmentation

Recognition in a continuous stream of data is challenging [6] and we believe continuous (dynamic) gesture recognition is even more challenging because wrong recognition could lead to wrong actions being performed in a gesture-based UI, causing user frustration. Additionally, continuous streams of data often contain non-gestural interactions which make the recognition problem more challenging, as a recognizer needs to correctly determine and ignore these non-important sub-parts.

Pavlovic *et al*. [84] identify three phases that comprise a dynamic gesture: *pre-stroke* (the movement in preparation for gesticulation), *nucleus* (the actual movement of the gesture) and *post-*

stroke (the movement to a resting pose). Determining the correct boundaries of each phase is the main task in gesture segmentation.

In broad strokes, Escalera *et al*. [85] categorize dynamic gesture recognition into two categories: *direct* and *indirect* methods. Direct methods include heuristic-based segmentation wherein low-level features (*e.g*. trajectory, velocity, *etc*.) are computed and a decision as to whether a gesture has occurred is made based on changes in these low level features. The popular approach of using a temporally sliding window along with a segmented gesture recognizer is another example of the direct methods. Conversely, indirect methods often perform recognition and segmentation jointly by analyzing the stream of data and determining the boundaries where a possible gesture may have been performed. In this work we concentrate on the indirect methods, as they typically do not rely on hand-engineered features and are often more practical in many application domains.

3.2.1 Traditional Methods

Traditionally, hidden Markov models (HMM) and conditional random fields (CRF) [86] have been used for such tasks, primarily because these probabilistic models inherently work well with sequential data. As Escalera *et al*. [85] point out, the use of a *threshold model* is common across probabilistic models used for gesture spotting or segmentation. In simple terms, the output of probabilistic models is often compared to a threshold to determine whether to accept the detection.

Inspired by natural language processing, Wang *et al*. [87] proposed a segmentation method based on HMMs for analyzing videos containing human gestures. Similarly, Yin and Davis [88] used salient features along with HMMs for hand gesture spotting and recognition. Sign language spotting has also been studied by Yang *et al*. [89]. In their method, they use CRFs along with a novel threshold model to not only recognize the components of the sign language, but also to distinguish between sign and non-sign gestures.

The use of DTW or continuous dynamic programming (CDP) [90] has also been explored for continuous gesture recognition. Alon *et al*. [91] proposed a unified framework that can be used for such recognition tasks. They demonstrated the merits of their framework via a DTW-based spatio-temporal matching algorithm capable of matching gestures to their exemplar templates in a continuous stream of data. More recently, Tang *et al*. [92] proposed a method of recognizing continuous hand gestures by combining a template- and velocity-based segmentation method with DTW.

All this prior work differs from ours in that we use deep networks which alleviate the need for hand-engineering features. Also, in our experience, deep networks are more flexible to use across various modalities and data representations.

3.2.2 Deep Networks

Given the ability of RNNs to work with sequence data, and their effectiveness in tasks such as speech recognition, it is unsurprising that these networks have been used for continuous recognition and segmentation.

One popular application of analyzing continuous sequences is sign language recognition, which has received a lot of attention in the literature. Koller *et al*. introduced Deep Sign [93], a hybrid model consisting of a CNN and a HMM for continuous sign language recognition. Their CNN was used for feature extraction whereas the HMM was used for sequence modeling. Yang *et al*. proposed SF-Net [94] which aimed to translate signing into a sequence, simultaneously solving recognition and sequence generation. Cui *et al*. developed a vision-based continuous sign language recognizer [95]. They used a model consisting of recurrent and convolutional units for feature extraction in both spatial as well as temporal dimensions to learn sequences.

Most of the existing work in the literature of continuous gesture recognition focus on analyzing

visual data, such as videos. Molchanov *et al*. introduced a multi-sensor continuous recognizer for gestures with a focus on applications in automotive interfaces [96]. Using a CNN, they fused the data from a short-range radar, a color and a time-of-flight depth sensor and recognized driver's hand gestures. Note that their method was not explicitly designed for online gesture recognition. Rather, they leveraged the existence of the radar data along with heuristics to determine when a gesture has begun or ended. Later Molchanov *et al*. [97] showed the merits of using a 3D CNN operating on spatial and temporal domains to recognize the driver's hand gestures from RGB-D video. Molchanov *et al*. extended the idea of using 3D CNNs along with RNNs to perform fully online gesture recognition [98]. Notably, their model used the connectionist temporal classification (CTC) [99] cost function as a means of identifying the nucleus of the gesture.

Other examples of continuous gesture recognition using visual data include the work of Zhu *et al*. [100]. They employed a multi-network architecture consisting of four models. One network performs the segmentation of the input, while three other networks work jointly to recognize the segmented gesture in isolation. The use of an attention model to improve continuous recognition has also been explored in the literature. Dhingra and Kunz introduced Res3ATN [101] an end-to-end attention network for hand gesture recognition. They studied the effect of varying the number of attention blocks on the recognition accuracy of the network.

Since most of such related work focuses on gesture recognition in visual data, they emphasize the use of CNNs as a means of extracting features from video sequences. Consequently, the application of these methods is limited to video data. In contrast, we put emphasis on the task of recognition or segmentation itself. Our goal is to present a general-purpose continuous recognizer that can be trained end-to-end and can work across various data and modalities.

3.3 Gesture Generation

Synthetic generation of data as pertinent to gesture recognition has been explored in the literature for a variety of reasons, such as lack of sufficient amounts of labeled data. Perhaps the most notable usage of such synthetic data was in Kinect [102], where large amounts of training data were needed for its pose recognition algorithm. Other examples include handwriting recognition of ancient texts [103], handwriting generation [104, 105] as well as digital forensics applications [106, 107]. As Taranta *et al*. [108] point out, an example of usage of synthetic gestures is for training recognizers with very little amounts of data, or to add the ability to customize gestures in an application.

In broad strokes, the synthetic generation methods often take two approaches. They either apply perturbations to existing samples and generate new sequences, or they build a generative model that can generate new samples without the need for inputting an existing one. Methods that generate new samples by means of geometrical transformations (*e.g*. scaling, rotating, *etc*.) fall into the former category. The latter category of methods often requires more data, but the generated instances are often of higher quality since these methods attempt to mimic the distribution of the original dataset of samples.

Notable examples of generation through perturbation of samples follow. Plamondon proposed the kinematic theory of rapid human movements [109] which describes human movements. Based on this, the Sigma-Lognormal model [110] was proposed as a means of generating handwriting strokes. Leiva *et al*. demonstrated the effectiveness of this model for gesture generation tasks [111, 112]. In their work, every gesture is modeled via the Sigma-Lognormal model which has six control parameters. New samples are generated by applying random perturbations on these parameters. Davila *et al*. [113] used *Perlin noise* [114] as the perturbation model for creating synthetic samples. Perlin noise is a computer graphics technique for generating textures. Taranta *et al*. introduced GPSR [108] which produces synthetic samples by stochastically resampling and

lengthening or shortening the gesture paths.

Examples of generative models often involve the use of neural networks. Prior to the advent of GANs [21], most sequence generation methods relied on *language modeling*, a probabilistic technique often used natural language processing and sequence prediction. Language modeling involves estimating the probability of a given sequence, considering a corpus of valid sequences. For instance, the English sentence *"The weather is good"* is much more probable than the sentence *"The wither is good"*. Thus, a language model trained on a corpus of English sentences is expected to assign a higher probability to the former. The extensive work of Graves [115] demonstrated the surprising effectiveness of such models in various sequence generation tasks, including handwriting generation.

More recently GAN-based methods have been used for various action and motion sequence generation tasks. Tang *et al*. [116] introduced GestureGAN, a generative model for hand gesture-to-gesture translation. GestureGAN takes two inputs: a single image of a person holding a static hand gesture, as well as a skeleton pose. It then produces as the output the image of that same person, but holding the gesture of the skeleton with the correct pose, thereby performing gesture translation. Yang *et al*. [117] presented a pose-guided human video generation method. The input to their framework consisted of an image of a person, along with an action label. Their goal was to generate a video of that person performing the specified action. Notably, their framework consisted of two separate GAN models, one for pose sequence generation and the other one for video generation based on the generated poses.

To our knowledge, prior work specifically focused on gesture sequence generation across different modalities is scarce, which makes our work novel. Most related to our work is the work of Zhang's *et al*. [118]. They used RNNs for Chinese character recognition and generation. Their generation was done through a GAN framework. Notably, they reused their proposed recognizer

as the discriminator in the GAN model, which is what we also do in Chapter 6. Also, they generate their characters in a temporal pen stroke format which is much more challenging than generating the image of characters. What sets us apart from the work of Zhang's *et al*. [118] is our explicit focus on gestures, which can include 2D pen strokes as well as 3D hand or full-body gestures. Moreover, as we show in Chapter 6 we formulate a novel loss function that allows us to generate gesture sequences with a single network, obviating the need for GAN models.

Training generative models without a discriminator has also been explored in the literature. Yu *et al*. [119] proposed generating text sequences using reinforcement learning techniques. Their work was later extended by Guo *et al*. [120] in which techniques from hierarchical reinforcement learning [121] was also incorporated. Lin *et al*. [122] presented the use of a ranking mechanism that replaced the discriminator of a GAN-like generator. Li *et al*. [123] introduced an adversarial optimization procedure to train a text generator. These prior work focus on generating sequences of discrete tokens which typically involve generating sequences comprised of tokens of a predetermined cardinality such as producing English text. Conversely, our goal is to generate real-valued and continuous multi-dimensional gesture sequences, which is highly challenging as data can take arbitrary values. Additionally, our loss formulation is different from [123] in that our formulation is non-adversarial in nature.

CHAPTER 4: SEGMENTED GESTURE RECOGNITION WITH RECURRENT NETWORKS USING DEEPGRU[1]

4.1 Introduction

In this chapter we introduce *DeepGRU*: our proposed end-to-end deep network-based gesture recognition utility (see Figure 4.1) designed for recognizing segmented gestures. DeepGRU works directly with the raw 3D skeleton, pose or other vector features (*e.g*. acceleration, angular velocity, *etc*.) produced by noisy commodity hardware, thus requiring minimal domain-specific knowledge to use. With roughly 4 million trainable parameters, DeepGRU is a rather small network by modern standards and is budget-aware when computational power is constrained. Through evaluations on different datasets and gesture modalities, we demonstrate that our proposed method achieves state-of-the-art recognition accuracy on small and large training data alike. We demonstrate the relevance of our deep network model for small-scale problems with a limited amount of training data. Specifically, we show that with as little as four training samples per class, our method can produce state-of-the-art results in such setting, and that it is possible to train our model in a reasonable amount of time using only the CPU.

4.2 DeepGRU

In this section we provide an in-depth discussion of DeepGRU's architecture. In our architecture, we take inspiration from VGG-16 [124], and the attention [12, 13] and sequence to sequence mod-

[1]Some portions of this chapter previously appeared in the following publication:
Maghoumi M., LaViola J.J. (2019) DeepGRU: Deep Gesture Recognition Utility. In: Bebis G. et al. (eds) Advances in Visual Computing. ISVC 2019. Lecture Notes in Computer Science, vol 11844. Springer, Cham
The final authenticated version is available online at https://doi.org/10.1007/978-3-030-33720-9_2

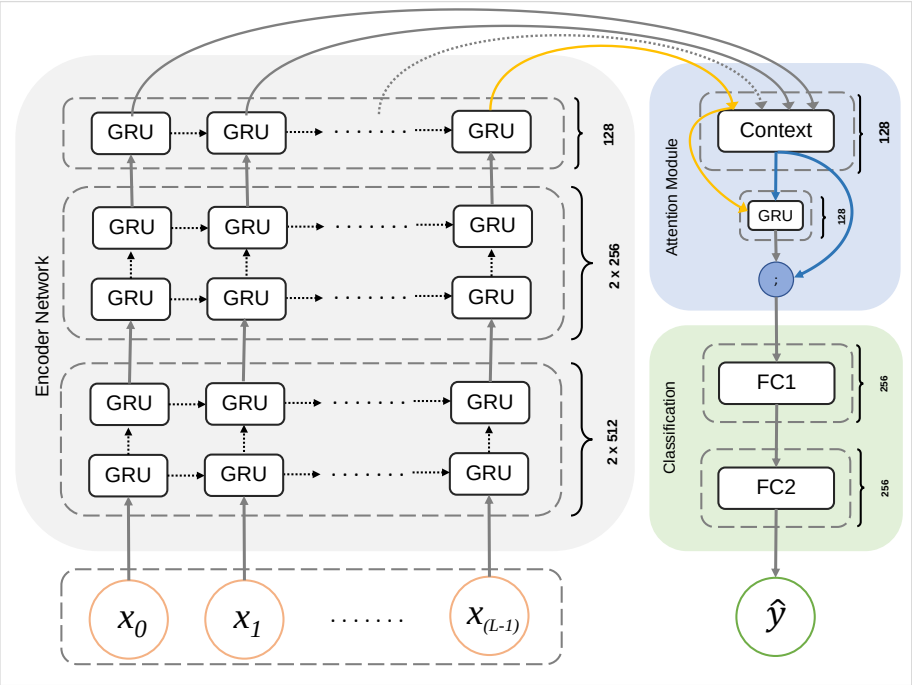


Figure 4.1: **DeepGRU** – the proposed recurrent model for gesture recognition which consists of an *encoder network* of stacked gated recurrent units (GRU), the *attention module* and the *classification* layers. The input $\mathbf{x} = (x_0, x_1, ..., x_{(L-1)})$ is a sequence of vector data of arbitrary length and the output is the predicted class label $\hat{y}$. The number of the hidden units for each layer is displayed next to every component (see Section 4.2 for a thorough description).

els [125]. Our model, depicted in Figure 4.1, is comprised of three main components: an encoder network, the attention module, and two fully-connected (FC) layers fed to softmax producing the probability distribution of the class labels. We provide an ablation study to give insight into our design choices in Section 4.4.

4.2.1 Input Data

The input to DeepGRU is raw input device samples represented as a temporal sequence of the underlying gesture data (*e.g.* 3D joint positions, accelerometer or velocity measurements, 2D Carte-

sian coordinates of pen/touch interactions, *etc*.). At time step t, the input data is the column vector $x_t \in \mathbb{R}^N$, where N is the dimensionality of the feature vector. Thus, the input data of the entire temporal sequence of a single gesture sample is the matrix $\mathbf{x} \in \mathbb{R}^{N \times L}$, where L is the length of the sequence in time steps. Note that the dimensionality N depends on the device that generated the data and also how one chooses to represent the data. In this sense, DeepGRU is agnostic to the input representation. Also, we use the entire temporal sequence as-is without sub-sampling or clipping.

4.2.2 Encoder Network

The encoder network in DeepGRU is fed with data from training samples and serves as the feature extractor. Our encoder network consists of a total of five stacked unidirectional GRUs. Although LSTM units [8] are more prevalent in the literature, we utilize GRUs because due to the smaller number of parameters, these units are simpler to use and are generally faster to train and are less prone to overfitting. At time step t, given an input vector x_t and the hidden state vector of the previous time step $h_{(t-1)}$, a GRU computes h_t, the hidden output at time step t, as $h_t = \mathbf{\Gamma}\big(x_t, h_{(t-1)}\big)$ using the transition equations detailed in Equation 2.3.

Given a gesture sample $\mathbf{x} \in \mathbb{R}^{N \times L}$, the encoder network uses Equation 2.3 to output $\bar{h} \in \mathbb{R}^{128 \times L}$, where $\bar{h}$ is the result of the concatenation $\bar{h} = \big[h_0;\ h_1;\ \ldots\ ;\ h_{(L-1)}\big]$. This output, which is a compact encoding of the input matrix $\mathbf{x}$, is then fed to the attention module.

4.2.3 Attention Module

The output of the encoder network, which is a compressed representation of the input gesture sample, can provide a reasonable set of features for performing classification. We further refine this set of features by extracting the most informative parts of the sequence using the attention

model. We propose a novel adaptation of the global attention model [13] which is suitable for our recognition task. Given all the hidden states $\bar{h}$ of the encoder network, our attention module computes the attentional context vector $c \in \mathbb{R}^{128}$ using the trainable parameters W_c as:

$$c = \text{softmax}\left(h_{(L-1)}^{\intercal} W_c \bar{h}\right) \bar{h} = \left(\frac{\exp\left(h_{(L-1)}^{\intercal} W_c \bar{h}\right)}{\sum_{t=0}^{L-1} \exp\left(h_{(L-1)}^{\intercal} W_c h_t\right)} \right) \bar{h} \tag{4.1}$$

As evident in Equation 4.1, we solely use the hidden states of the encoder network to compute the attentional context vector. The hidden state of the last time step $h_{(L-1)}$ of the encoder network (the yellow arrow in Figure 4.1) is the main component of our context computation and attentional output. This is because $h_{(L-1)}$ can potentially capture a lot of information from the entire gesture sample sequence.

With the context vector in hand, one could use the concatenation $\left[c\ ;\ h_{(L-1)}\right]$ to form the contextual feature vector and perform classification. However, recall that the inputs to DeepGRU can be of arbitrary lengths. Therefore, the amount of information that is captured by $h_{(L-1)}$ could differ among short sequences and long sequences. This could make the model susceptible to variations in sequence lengths. Our proposed solution to mitigate this is as follows. During training, we jointly learn a set of parameters that given the context and the hidden state of the encoder network would decide whether to use the hidden state directly, or have it undergo further transformation while accounting for the context. This decision logic can be mapped to the transition equations of a GRU (see Equation 2.3). Thus, after computing the context c, we additionally compute the auxiliary context c' and produce the attention module's output o_{attn} using $\mathbf{\Gamma}_{\text{attn}}$ (the attentional GRU) as follows:

$$
\begin{aligned}
c' &= \mathbf{\Gamma}_{\text{attn}}\big(c, h_{(L-1)}\big) \\
o_{\text{attn}} &= \big[c\ ;\ c'\big]
\end{aligned}
\tag{4.2}
$$

In summary, we believe that the novelty of our attention model is threefold. First, it only relies on the hidden state of the last time step $h_{(L-1)}$, which reduces complexity. Second, we compute the auxiliary context vector to mitigate the effects of sequence length variations. Lastly, our attention module is invariant to zero-padded sequences and thus can be trivially vectorized for training on mini-batches of sequences with different lengths. As we show in Section 4.4, our attention model works very well in practice.

4.2.4 Classification

The final layers of our model are comprised of two FC layers (F_1 and F_2) with ReLU activations that take the attention module's output and produce the probability distribution of the class labels using a softmax classifier:

$$
\hat{y} = \text{softmax}\Big(\text{F}_2\Big(\text{ReLU}\big(\text{F}_1(o_{\text{attn}})\big)\Big)\Big) \tag{4.3}
$$

We use batch normalization [126] followed by dropout [127] on the input of both F_1 and F_2 in Equation 4.3. During training, we minimize the cross-entropy loss to reduce the difference between predicted class labels $\hat{y}$ and the ground truth labels y.

4.3 Evaluation

To demonstrate the robustness and generality of DeepGRU, we performed a set of experiments on datasets of various sizes. Specifically, we evaluate our proposed method on five datasets: UT-Kinect [3], NTU RGB+D [77], SYSU-3D [51], DHG 14/28 [4, 128] and SBU Kinect Interactions [129]. We believe these datasets cover a wide range of gesture interactions, number of actors, viewpoint variations and input devices. We additionally performed experiments on two small-scale datasets (Wii Remote [55] and Acoustic [83]) in order to demonstrate the suitability of DeepGRU for scenarios where only a very limited amount of training data is available. We compute the recognition accuracies on each dataset and report them as a percentage.

Implementation details. We implemented DeepGRU using the PyTorch [11] framework. The input data to the network are z-score normalized using the training set. We use the Adam solver [7] ($\beta_1 = 0.9, \beta_2 = 0.999$) and the initial learning rate of 10^{-3} to train our model. The mini-batch size for all experiments is 128, except for those on NTU RGB+D, for which the size is 256. Training is done on a machine equipped with two NVIDIA GeForce GTX 1080 GPUs, Intel Core-i7 6850K processor and 32 GB RAM. Unless stated otherwise, both GPUs were used for training with mini-batches divided among both cards. Our reference implementation is available at *https://github.com/Maghoumi/DeepGRU*.

Regularization. We use dropout (0.5) and data augmentation to avoid overfitting. All regularization parameters were determined via cross-validation on a subset of the training data. Across all experiments we use three types of data augmentation: **(1)** random scaling with a factor[2] of ± 0.3, **(2)** random translation with a factor of ± 1, **(3)** synthetic sequence generation with gesture path

[2] A factor of ± 0.3 indicates that samples are randomly and non-uniformly (*e.g.*) scaled along all axes to [0.7, 1.3] of their original size.

stochastic resampling (GPSR) [108]. For GPSR we randomly select the resample count n and remove count r. We use n with a factor of ($\pm 0.1 \times \widetilde{\mathrm{L}}$) and r with a factor of ($\pm 0.05 \times \widetilde{\mathrm{L}}$). Additionally, we use two more types of regularization for experiments on NTU RGB+D dataset. We use a weight decay value of 10^{-4}, as well as random rotation with a factor of $\pm\frac{\pi}{4}$. This was necessary due to the multi-view nature of the dataset.

4.3.1 UT-Kinect

This dataset [3] is comprised of ten gestures performed by ten participants two times (200 sequences in total). The data of each participant is recorded and labeled in one continuous session. What makes this dataset challenging is that the participants move around the scene and perform the gestures consecutively. Thus, samples have different starting positions and/or orientations. We use the leave-one-out-sequence cross validation protocol of [3]. During our tests, we noticed that the label of one of the sequences was corrupted[3]. We manually labeled the sequence and performed our experiments twice: once with the corrupted sequence omitted, and once with our manually labeled version of the corrupted sequence. We obtained the same results in both settings. We achieve state-of-the-art results with the perfect classification accuracy of 100% as shown in Table 4.1.

4.3.2 NTU RGB+D

To our knowledge, this is the largest dataset of actions collected from Kinect (v2) [77]. It comprises about 56,000 samples of 60 action classes performed by 40 subjects. Each subject's skeleton has 25 joints. The challenging aspect of this dataset stems from the availability of various viewpoints for each action, as well as the multi-person nature of some action classes. We follow the cross-

[3]The second sample of participant 10's *carry* gesture

Table 4.1: Results on UT-Kinect [3] dataset.

Method	**Accuracy**
Grassmann Manifold [53]	88.5
Histogram of 3D Joints [3]	90.9
Riemannian Manifold [49]	91.5
Key-Pose-Motifs [130]	93.5
LARP + mfPCA [131]	94.8
Action snippets [132]	96.5
ST LSTM + Trust Gates [79]	97.0
Lie Group [58]	97.1
Graph-based [59]	97.4
ST-NBNN [56]	98.0
SCK + DCK [133]	98.2
DPRL + GCNN [134]	98.5
GCA-LSTM *(direct)* [19]	98.5
CNN + Kernel Feature Maps [135]	98.9
GCA-LSTM *(step-wise)* [19]	99.0
CNN + LSTM [81]	99.0
KRP FS [136]	99.0
DeepGRU	**100.0**

subject (CS) and cross-view (CV) evaluation protocols of [77]. In the CS protocol, 20 subjects are used for training and the remaining 20 subjects are used for testing. In the CV protocol, two viewpoints are used for training and the remaining one viewpoint is used for testing. Note that according to the dataset authors, 302 samples in this dataset have missing or incomplete skeleton data which were omitted in our tests.

We create our feature vectors similar to [77]. For each action frame, we concatenate the 3D coordinates of the skeleton joints into one 75 dimensional vector in the order that they appear in the dataset. In cases where there are multiple skeletons in a single action frame, we treat each skeleton as one single time step. For each frame, we detect the main actor, which is the skeleton with the largest amount of total skeleton motion. The time step frames are created in descending

Table 4.2: Results on NTU RGB+D [77] dataset.

Modality	**Method**	**Accuracy**	
		CS	CV
Image	Multitask DL [137]	84.6	–
	Glimpse Clouds [17]	**86.6**	**93.2**
Pose+Image	DSSCA - SSLM [138]	74.9	–
	STA Model (Hands) [139]	82.5	88.6
	Hands Attention [16]	84.8	90.6
	Multitask DL [137]	**85.5**	–
Pose	Skeletal Quads [61]	38.6	41.4
	Lie Group [58]	50.1	52.8
	HBRNN [140]	59.1	64.0
	Dynamic Skeletons [51]	60.2	65.2
	Deep LSTM [77]	60.7	67.3
	Part-aware LSTM [77]	62.9	70.3
	ST LSTM + Trust Gates [79]	69.2	77.7
	STA Model [20]	73.2	81.2
	LSTM + FA + VF [18]	73.8	85.9
	Temporal Sliding LSTM [80]	74.6	81.3
	CNN + Kernel Feature Maps [135]	75.3	–
	SkeletonNet [75]	75.9	81.2
	GCA-LSTM *(direct)* [19]	74.3	82.8
	GCA-LSTM *(step-wise)* [19]	76.1	84.0
	JTM CNN [141]	76.3	81.1
	DPTC [82]	76.8	84.9
	VA-LSTM [78]	79.4	87.6
	Beyond Joints [142]	79.5	87.6
	Clips+CNN+MTLN [143]	79.6	84.8
	View-invariant [144]	80.0	87.2
	Dual Stream CNN [145]	81.1	87.2
	DPRL + GCNN [134]	83.5	89.8
	DeepGRU	**84.9**	**92.3**

order of total skeleton motion. Following [77], we transform the coordinates of all skeletons to the spine-mid joint of the main actor in the action frame.

Our results are presented in Table 4.2. Although DeepGRU only uses the raw skeleton positions of the samples, we present the results of other recognition methods that use other types of gesture

Table 4.3: Results on SYSU-3D [51].

Method	**Accuracy**
LAAF [52]	54.2
Dynamic Skeletons [51]	75.5
ST LSTM + Trust Gates[79]	76.5
DPRL + GCNN [134]	76.9
VA-LSTM [78]	77.5
GCA-LSTM *(direct)* [19]	77.8
GCA-LSTM *(step-wise)* [19]	78.6
DeepGRU	**80.3**

data. To the best of our knowledge, DeepGRU achieves state-of-the-art performance among all methods that only use raw skeleton pose data.

4.3.3 SYSU-3D

This Kinect-based dataset [51] contains 12 gestures performed by 40 participants totaling 480 samples. The widely-adopted evaluation protocol [51] of this dataset is to randomly select 20 subjects for training and the use remaining 20 subjects for testing. This process is repeated 30 times and the results are averaged. The results of our experiments are presented in Table 4.3.

4.3.4 DHG 14/28

This dataset [128] contains 14 hand gestures of 28 participants collected by a near-view Intel RealSense depth camera. Each gesture is performed in two different ways: using the whole hand, or just one finger. Also, each sample gesture is repeated between one to ten times yielding 2800 sequences. The training and testing data on this dataset are predefined and evaluation can be performed in two ways: classify 14 gestures or classify 28 gestures. The former is insensitive to

Table 4.4: Results on DHG 14/28 [128] with two evaluation protocols.

Protocol	**Method**	**Accuracy**	
		C = 14	C = 28
Leave-one-out	Chen *et al*. [60]	84.6	80.3
	De Smedt *et al*. [146]	82.5	68.1
	CNN+LSTM [81]	85.6	81.1
	DPTC [82]	85.8	80.2
	DeepGRU	**92.0**	**87.8**
SHREC'17 [4]	HOG^2 [147][4]	78.5	74.0
	HIF3D [148]	90.4	80.4
	De Smedt *et al*. [149][4]	88.2	81.9
	Devineau *et al*. [76]	91.2	84.3
	DLSTM [74]	**97.6**	**91.4**
	DeepGRU	94.5	**91.4**

how an action is performed, while the latter discriminates the samples performed with one finger from the ones performed with the whole hand. The standard evaluation protocol of this dataset is a leave-one-out cross-validation protocol. However, the SHREC 2017 [4] challenge introduces a secondary protocol in which training and testing sets are pre-split. Table 4.4 depicts our results using both protocols and both number of gesture classes.

4.3.5 SBU Kinect Interactions

This dataset [129] contains 8 two-person interactions of seven participants. We utilize the 5-fold cross-validation protocol of [129] in our experiments. Contrary to other datasets, which express joint coordinates in the world coordinate system, this dataset has opted to normalize the joint values instead. Despite using a Kinect (v1) sensor, the participants in the dataset have only 15 joints.

We treat action frames that contain multiple skeletons similarly to what we described above for

Table 4.5: Results on SBU Kinect Interactions [129].

Modality	**Method**	**Accuracy**
Image	Hands Attention [16]	72.0
	DSPM	93.4
Pose + Image	Hands Attention [16]	94.1
Pose	HBRNN [140]	80.4
	Deep LSTM [77]	86.0
	Ji *et al*. [150]	86.8
	Co-occurance Deep LSTM [151]	90.4
	Hands Attention [16]	90.5
	STA Model [20]	91.5
	ST LSTM + Trust Gates [79]	93.3
	SkeletonNet [75]	93.5
	Clips + CNN + MTLN [143]	93.5
	GCA-LSTM *(direct)* [19]	94.1
	CNN + Kernel Feature Maps [135]	94.3
	GCA-LSTM *(step-wise)* [19]	94.9
	LSTM + FA + VF [18]	95.0
	VA-LSTM [78]	**97.2**
	DeepGRU	95.7

the NTU RGB+D dataset, with the exception of transforming the joint coordinates. Also, using the equations provided in the datasets, we covert the joint values to metric coordinates in the depth camera coordinate frame. This is necessary to make the representation consistent with other datasets that we experiment on. Table 4.5 summarizes our results.

4.3.6 Small Training Set Evaluation

The amount of training data for some gesture-based applications may be limited. This is especially the case during application prototyping stages, where developers tend to rapidly iterate through design and evaluation cycles. Throughout the years, various methods have been proposed in the

literature aiming to specifically address the need for recognizers that are easy to implement, fast to train and work well with small training sets [44, 45, 71, 152].

Traditionally, deep networks are believed to be slow to train, requiring a lot of training data. We show this is not the case with DeepGRU and our model performs well with small training sets and can be trained only on the CPU. We pit DeepGRU against Protractor3D [71], $3 [152] and Jackknife [44] which to our knowledge produce high recognition accuracies with a small number of training instances [44].

We examine two datasets. The first dataset contains acoustic over-the-air hand gestures via Doppler shifted soundwaves [83]. This dataset contains 18 hand gestures collected from 22 participants via five speakers and one microphone. At 165 component vectors per frame, this dataset is very high-dimensional. Also, the soundwave-based interaction modality is prone to high amounts of noise. The second dataset contains gestures performed via a Wii Remote controller [55]. This dataset contains 15625 gestures of 25 gesture classes collected from 25 participants. In terms of data representation, both datasets differ from all others examined thus far. Samples of [83] are frequency binned spectrograms while samples of [55] are linear acceleration data and angular velocity readings (6D), neither of which resemble typical skeletal representations nor positional features.

For each experiment we use the user-dependent protocol of [44, 55]. Given a particular participant, $\mathcal{T}$ random samples from that participant are selected for training and the remaining samples are selected for testing. This procedure is repeated per participant and the results are averaged across all trials. Considering that in the prototyping stage the amount of training samples is typically limited, we evaluate the performance of all the recognizers using $\mathcal{T}$=2 and $\mathcal{T}$=4 training samples per gesture class. The results are tabulated in Table 4.6. We see that with $\mathcal{T}$=4 training samples per gesture class, DeepGRU outperforms other recognizers on both datasets.

Table 4.6: Rapid prototyping evaluation results with T training samples per gesture class.

Dataset	**Method**	**Accuracy**	
		$\mathcal{T} = 2$	$\mathcal{T} = 4$
Acoustic [83]	**Jackknife** [44]	**91.0**	94.0
	DeepGRU	89.0	**97.4**
Wii Remote [55]	Protractor3D [71]	73.0	79.6
	$3 [152]	79.0	86.1
	Jackknife [44]	**96.0**	98.0
	DeepGRU	92.4	**98.3**

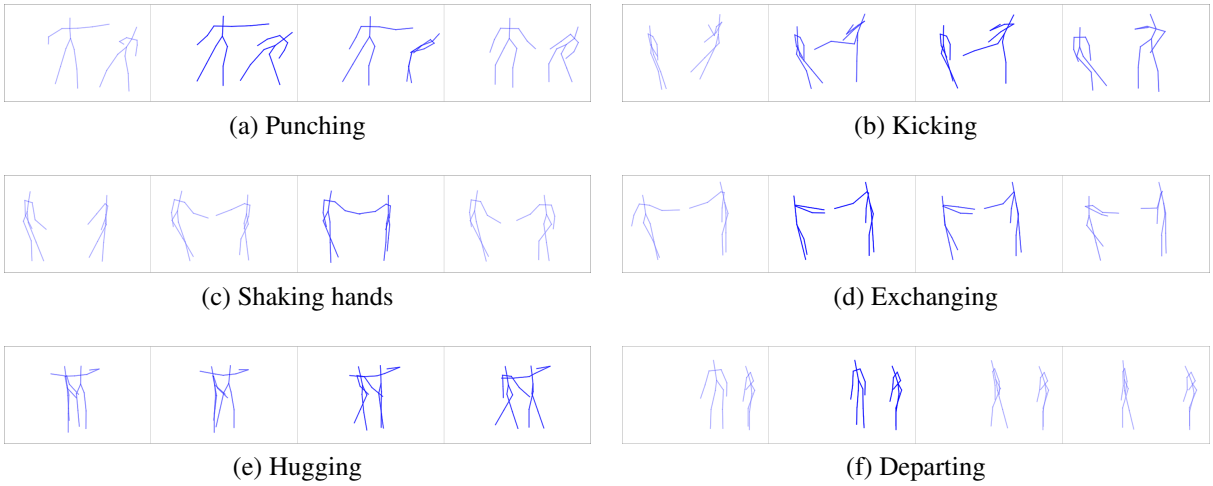
Table 4.7: DeepGRU training times (in minutes) on various datasets.

Device	**Configuration**	**Dataset**	**Time (mins)**
CPU	12 threads	Acoustic [83] ($\mathcal{T}$=4)	1.7
		Wii Remote [55] ($\mathcal{T}$=4)	6.9
GPU	2× GTX 1080	SHREC 2017 [4]	5.5
		NTU RGB+D [77]	129.6
	1× GTX 1080	SHREC 2017 [4]	6.2
		SYSU-3D [51]	9.0
		NTU RGB+D [77]	198.5

4.4 Discussion

Comparison with the state-of-the-art. Experiment results show that DeepGRU generally tends to outperform the state-of-the-art results, sometimes with a large margin. On the NTU-RGB+D [77], we observe that in some cases DeepGRU outperforms image-based or hybrid methods.

Although the same superiority is observed on the SBU dataset [129], our method achieves slightly lower accuracy compared to VA-LSTM [78]. One possible intuition for this observation could be that the SBU dataset [129] provides only a subset of skeleton joints that a Kinect (v1) device can produce (15 compared to the full set of 20 joints). Further, note that VA-LSTM's view-adaptation



(a) Punching (b) Kicking

(c) Shaking hands (d) Exchanging

(e) Hugging (f) Departing

Figure 4.2: Attention response visualization for samples from the SBU Kinect Interactions [129] dataset. Each sample starts from the left and progresses to the right. The color intensity indicates the amount of attentional response (norm) to the frame (darker = higher response).

sub-network assumes that the gesture data are 3D positions and viewpoint-dependent. This is in contrast with DeepGRU which does not make such assumptions about the underlying type of the input data (position, acceleration, velocity, *etc*.).

As shown in Table 4.4, classifying 14 gestures of the DHG 14/28 dataset [128] with DLSTM [74] yields higher recognition accuracy compared to DeepGRU. As previously mentioned, DLSTM [74] uses hand-crafted angular features extracted from hand joints and these features are used as the input to the recurrent network while DeepGRU uses raw input, which relieves the user of the burden of computing domain-specific features. Classifying 28 classes, however, yields similar results with either of the recognizers.

Generality. Our experiments demonstrate the versatility of DeepGRU for various gesture or action modalities and input data: from full-body multi-actor actions to hand gestures, collected from various commodity hardware such as depth sensors or game controllers with various data represen-

tations (*e.g.* pose, acceleration and velocity or frequency spectrograms) as well as other differences such as the number of actors, gesture lengths, number of samples and number of viewpoints. Regardless of these differences, DeepGRU can still produce high accuracy results.

This flexibility is, in large part, due to our attention module and context vector computation. We present an example visualization of our attention module's response in Figure 4.2. We can see that after training, our attention mechanism correctly selects the most discriminative frames in the sequence. Additionally, we did not observe much overfitting during training suggesting that our regularization methods seem to generalize well to various application domains.

Ease of use. In addition to accuracy, the adoption of any one gesture recognition method ultimately comes down to the ease of use. In that regard, DeepGRU has a few advantages over competitive methods. Our method uses raw device data, thus requiring fairly little domain knowledge. Our model is straightforward to implement and as we discuss shortly, training is fast. We believe these traits make DeepGRU an enticing option for practitioners.

Timings. Training time is an important factor in the prototyping stage. In such scenarios, the ability to conveniently train a network without GPUs is desirable. We measured the amount of time it takes to train DeepGRU to convergence with different configurations in Table 4.7. The reported times include dataset loading, preprocessing and data augmentation time. Training our model to convergence tends to be fast. In fact, GPU training of medium-sized datasets or CPU-only training of small datasets can be done in under 10 minutes, which we believe is beneficial for iterative design. We also measured DeepGRU's average inference time per sample both on GPU and on CPU in *micro*seconds. On a single GPU, our method takes $349.1\,\mu s$ to classify one gesture sample while it takes $3136.3\,\mu s$ on the CPU.

Ablation study. To provide insight into our network design, we present an ablation study in Ta-

Table 4.8: Ablation study on DHG 14/28 dataset (14 class, SHREC'17 protocol). We examine (respectively) the effects of the usage of the attention model, the recurrent layer choice (LSTM *vs*. GRU), the number of stacked recurrent layers (3 *vs*. 5) and the number of FC layers (1 *vs*. 2). Training times (seconds) are reported for every model. Experiments use the same random seed. DeepGRU's model is boldfaced.

Attn.	Rec. Unit	# Stacked	# FC	Time (sec)	Accuracy	Attn.	Rec. Unit	# Stacked	# FC	Time (sec)	Accuracy
-	LSTM	3	1	162.21	91.78	✓	LSTM	3	1	188.29	92.74
-	LSTM	3	2	164.07	91.07	✓	LSTM	3	2	192.12	92.02
-	LSTM	5	1	246.47	91.90	✓	LSTM	5	1	277.32	92.38
-	LSTM	5	2	251.67	89.52	✓	LSTM	5	2	283.35	92.26
-	GRU	3	1	143.87	93.45	✓	GRU	3	1	170.48	94.12
-	GRU	3	2	148.08	93.33	✓	GRU	3	2	174.00	93.81
-	GRU	5	1	210.83	93.69	✓	GRU	5	1	243.10	93.93
-	GRU	5	2	212.99	93.81	✓	**GRU**	**5**	**2**	**248.66**	**94.52**

ble 4.8. Most importantly, we note depth alone is not sufficient to achieve state-of-the-art results. Further, accuracy increases in all cases when we use GRUs instead of LSTMs. GRUs were on average 12% faster to train and the worst GRU variant achieved higher accuracy than the best LSTM one. In our early experiments, we noted LSTM networks overfitted frequently which necessitated a lot more parameter tuning, motivating our preference for GRUs. However, we later observed underfitting when training GRU variants on larger datasets, arising the need to reduce regularization and tune parameters again. To alleviate this, we added the second FC layer which later showed to improve results across all datasets while still faster than LSTMs to train. We observe increased accuracy in all experiments with attention, which suggests the attention model is necessary. Lastly, in our experiments we observed an improvement of roughly 0.5%–1% when the auxiliary context vector is used (Section 4.2.3). In short, we see improved results with the attention model on GRU variants with five stacked layers and two FC layers.

Limitations. Our method has some limitations. Most importantly, the input needs to be segmented, although adding support for unsegmented data is straightforward, as we discuss in the next chapter. In our experiments we observed that DeepGRU typically performs better with high-

dimensional data, thus application on low-dimensional data may require further effort from developers. Although we used a similar set of hyperparameters for all experiments, other datasets may require some tuning. Also, the data augmentation methods that we used for our experiments have various hyperparameters and depending on the application, one may need to manually tune these hyperparameters.

4.5 Conclusion

We discussed DeepGRU, a deep network-based gesture and action recognizer which directly works with raw pose and vector data. We demonstrated that our architecture, which uses stacked GRU units and a global attention mechanism along with two fully-connected layers, was able to achieve state-of-the-art recognition results on various datasets, regardless of the dataset size and interaction modality. We further examined our approach for application in scenarios where training data is limited and computational power is constrained. Our results indicate that with as little as four training samples per gesture class, DeepGRU can still achieve competitive accuracy. We also showed that training times are short and CPU-only training is possible. We believe these properties make our proposed method a viable option for rapid prototyping and development scenarios.

CHAPTER 5: CONTINUOUS GESTURE RECOGNITION WITH RECURRENT NETWORKS USING UDEEPGRU

5.1 Introduction

We introduced DeepGRU in Chapter 4, a deep learning-based recognizer for action and gesture recognition of samples collected from commodity input devices. One of the limitations of DeepGRU was that it was designed to work with segmented data. We extend DeepGRU and apply it to the task of recognizing continuous (unsegmented) gestures in an online recognition scenario. Recognition of continuous gestures is an exciting field as it opens the doors for realizing many fluid and novel UI interactions. Project Soli [153] is one recent example of how continuous gesture recognition can be used to greatly enhance user experience.

5.2 Task Definition

In a continuous recognition scenario, gestures are fed into the system as a continuous stream of data, simulating an interaction environment in which the users are free to use the interface arbitrarily. Users may or may not decide to invoke gestural commands. Consequently, the boundaries of each gesture are not demarcated, and no prior knowledge about the existence of any particular gesture is present. One definition for recognition in such scenarios would be spotting the valid gestures in that continuous stream of data: the system outputs a class label as soon as it determines the corresponding gesture was performed by the user. Alternatively, one could pose the recognition process as a segmentation problem in which the goal is to determine what (if any) gesture was performed, and where the starting and ending points were. This is equivalent to labeling every time step of the data, and extracting each segment of gestures based on the boundaries of their

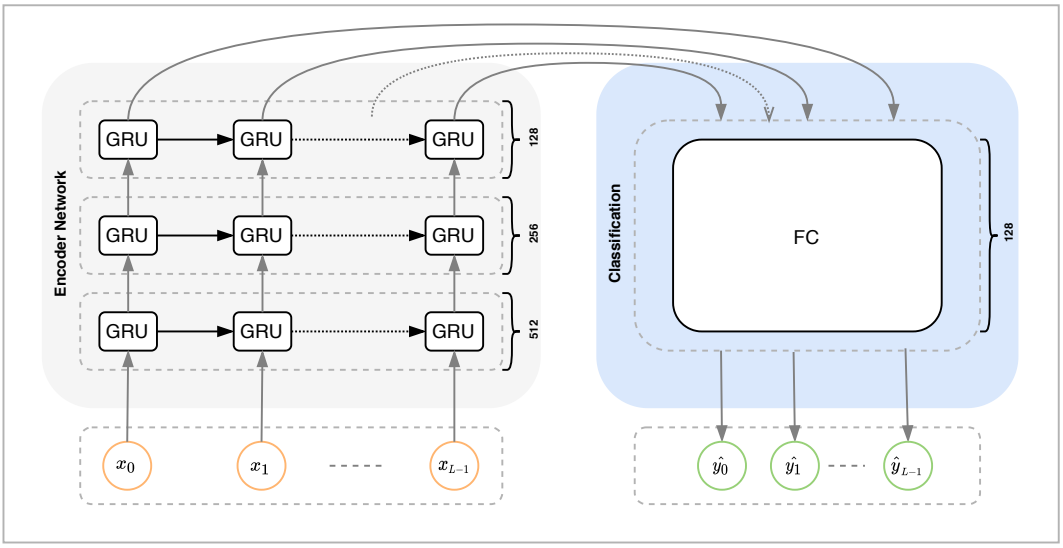


Figure 5.1: **uDeepGRU** – the adaptation of DeepGRU for continuous gesture recognition. uDeepGRU consists of an encoder network and a classification sub-network

occurrence. Due to these additional requirements, segmentation is a more challenging task [6]. In this chapter, we focus on the recognition of continuous gestures through segmentation.

5.3 uDeepGRU

We refer to the variation of DeepGRU for unsegmented gesture recognition as *uDeepGRU*. Figure 5.1 depicts our network's architecture, which is very similar in spirit to DeepGRU. Our network consists of the encoder and the classification sub-networks. We use unidirectional GRUs as the recurrent layers of our encoder network with the same transition equations as DeepGRU (see 2.3). The output of every GRU unit depends on all previously observed time steps only, making the network suitable for use in online recognition applications. The classification sub-network consists of one fully connected layer, preceded by batch normalization [126] and dropout [127]. Contrary to DeepGRU, uDeepGRU is a shallower network and does not include an attention [12] sub-network.

These omissions were motivated by our ablation study (see Table 4.8), and were done to reduce the size of the network without drastically affecting its recognition power.

uDeepGRU works very similar to DeepGRU. The input to the network is the vector $\mathbf{x}$, the raw input device samples represented as a temporal sequence of the underlying gesture data. Thus, a N dimensional feature vector of length L implies $\mathbf{x} \in \mathbb{R}^{N \times L}$. Our goal is to label each frame $x_t \in \mathbf{x}$ with the correct class label. Note that because the input data at time step t may not contain any gestures, we need the ability to output "no gesture" specifiers too. As such, we treat no-gestures as their own class.

More formally, given a dataset with gesture classes $C = \{c_0, c_1, ..., c_c\}$, we define the augmented class of gestures $\widetilde{C} = \{\text{None} \cup C\}$, and define our goal as labeling each feature vector (frame) x_t with a class label $\hat{y}_t$ where $\hat{y}_t$ is the predicted label of the input frame at time step t computed from the class-conditional probability estimate $P(\hat{y}_t|x_t)$.

5.3.1 Training Loss

As stated previously, given a feature vector x_t at each time step, the network outputs $\hat{y}_t$, the predicted label of the input frame at time step t. To reduce the difference between predicted class labels $\hat{y}$ and the ground truth labels y during training, we minimize the unweighted sum of two losses, namely the cross-entropy loss and the negative Sorensen-Dice coefficients (commonly known as the Dice loss [154]). The Sorensen-Dice coefficients between two sets X and Y is defined as:

$$dice(X, Y) = \frac{2|X \cap Y|}{|X| + |Y|} \tag{5.1}$$

where $|X|$ is the cardinality of the set X. Minimizing the negative Dice loss is equivalent to

maximizing the F_1 score computed between per-frame predicted and ground truth labels, yielding improved overall recognition performance for tasks in which high precision and recall metrics is desired.

5.4 SHREC 2019 Track: Online Gesture Recognition

We participated in the SHREC 2019 Online Gesture Recognition challenge [1] with uDeepGRU, and took first place by achieving the top recognition accuracy among other participants. Here, we overview the results of the competition.

5.4.1 Task Definition

The goal of this challenge was for teams to come up with accurate continuous gesture recognizers that could recognize the class of the articulated gesture, and annotate the start and end of each gesture. One requirement for participants was that their methods should simulate recognition in an online recognition scenario; *i.e*. only rely on the data stream seen thus far. This implies the decision on the input vector $\mathbf{x}_t$, should solely be based on the previously seen feature vectors $\{\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_{t-1}\}$.

The dataset of this challenge was a set of hand gestures performed by 13 participants collected from a leap motion device. There are a total of five gesture classes. During gesture articulations, The full position and orientation of the hand joints were recorded. A total of 195 samples were collected, of which 60 samples from 4 subjects were annotated and provided to the participants as the training data. The remaining 135 samples were reserved for testing and their annotations were not publicly released.

The evaluation was done by measuring the recognition accuracy, as well as the correctness of the estimated gesture start and end times.

5.4.2 Training uDeepGRU

Considering the requirements for this challenge, and that the amount of training data was rather limited, we carried out some additional data pre-processing to improve our odds of winning.

We treated each frame of the data as one 48-dimensional vector $\mathbf{x}_t$, containing the concatenated 3D position of all joints, exactly as they appeared in data files, for time step t. We z-score normalized every data using the mean and standard deviation of the training set and represented every sample as trajectory paths $\vec{\mathbf{x}}$, which was the data that was fed to our network. We experimentally found the benefit of using trajectory paths over raw point representation.

The training data for this challenge was limited. We overcame this limitation by employing synthetic data generation techniques to augment and expand the dataset. Namely, we employed four complementary techniques: gesture path stochastic resampling (GPSR) [108], Fourier coefficient perturbations, time-series inversion, and rotations. First, GPSR uses a two-step approach to generate synthetic variations of a given trajectory, where one samples random points along the trajectory and then normalizes the distance between each pair of consecutive points. This process changes the velocity profile of a sample such that changes in curvature are temporally unique relative to the original input, thereby resulting in a unique shape. Second, since GPSR does not significantly warp low frequency information (straight edges will remain straight), we further modify the trajectory's Fourier coefficients. That is, we perform a discrete Fourier transform (DFT) on a given trajectory, randomly adjust the amplitude of each coefficient by $\pm 20\%$, and then perform an inverse DFT to synthesize a variant of the original trajectory. The effect of this transformation is that a given sample will have "wobbles" not previously embedded within the trajectory (*e.g.* previously straight

edges may now have some bend). Note that results will also have additional high frequency noise that one can remove with a low pass filter; however in our testing, we found this noise was unharmful. Third, in order to increase the amount of non-gesture training data available to our system, we reversed each training sample's trajectory. Finally, we also added random rotations to each trajectory in order to increase orientation variance. Specifically, we randomly rotated trajectories ± 10 degrees around the x-axis. Using these techniques, we generated five new synthetic samples per each real sample and trained uDeepGRU using all available data. We withheld 10% of the original training data for validation and trained the network end-to-end.

We implemented uDeepGRU using the PyTorch [11] framework. We used the Adam solver [7] ($\beta_1 = 0.9, \beta_2 = 0.999$) and the initial learning rate of 10^{-3} to train our model. The mini-batch size for our experiments was 128. Training was done on a machine equipped with one NVIDIA GeForce GTX 1080 GPUs, Intel Core-i7 6850K processor and 32 GB RAM. We saved the model that produced the best F_1 score on the validation set.

5.4.3 Obtaining Recognition

At test time, we ran each test sample through the network and obtain per-frame class labels $\hat{y}_t$. The outputs were optionally post-processed by **(1)** thresholding the class-conditional probabilities based on a predefined threshold $\mathbb{T}$, and **(2)** ensuring that at least $\mathbb{N}$ frames before every frame were assigned the same output label.

We obtained 3 sets of results. The first set of the result was the unprocessed output of our network obtained after applying the best-performing model on the supplied test data. The second set consisted of the results of an ensemble of six networks, each trained on different folds of the training set. The outputs of the models were averaged and post-processed with $\mathbb{T}$=0.5, $\mathbb{N}$=5 before the final labels were generated. The last set of experiments contained the results of the first set, plus post

processing applied to the raw outputs with $\mathbb{T}$=0.5, $\mathbb{N}$=1.

5.4.4 Contenders

In this section, we briefly discuss the methods that uDeepGRU was pitted against.

Basline. The baseline method was the sliding window 3-cent recognizer [73], which is a k-nearest neighbor recognizer.

PI-RN and AJ-RN. Two recurrent network models, consisting of LSTM units and fully-connected layers. One neural network used palm and index finger features, while the other used all joint information.

DeA: Divide et Agnosce. This model consisted of two different networks, one for segmentation and the other for recognition. The segmentation network employed LSTM units with softmax activation the goal of which was to determine when any gesture started and ended. The recognition of the spotted gesture was delegated to the recognizer network, consisting of LSTM units and fully-connected layers.

Segment LSTM. Which was yet another LSTM-based network. The distinguishing characteristic of this method was the use of crops in the length of 20 frames to increase recognition accuracy.

5.4.5 Results

Two sets of evaluations were performed and reported in SHREC 2019. These sets were chosen as they reflect the most pertinent measure to online recognition tasks, namely recognition accuracy (Table 5.1) and latency (Table 5.2).

Table 5.1: Recognition results of SHREC 2019 in percentage.

Method	**Correct**	**Mislabeled**	**False Detections**	**Missed**
uDeepGRU$_2$	85.2	7.4	3.0	4.4
uDeepGRU$_1$	79.3	8.1	3.0	9.6
uDeepGRU$_3$	79.3	8.1	2.2	10.4
3-cent	75.6	16.3	2.2	5.9
DeA	51.9	18.5	25.2	4.4
AJ-RN	28.1	43.0	23.0	5.9
PI-RN	11.1	39.3	48.9	0.7
Segment LSTM$_1$	11.1	28.9	60.0	0.0
Segment LSTM$_2$	6.7	25.2	68.1	0.0

Table 5.2: Latency results of SHREC 2019, in terms of the offset from the average gesture start/end positions (in seconds).

Method	**Start Offset**	**End Offset**
uDeepGRU$_2$	0.66	-1.66
uDeepGRU$_1$	0.21	-2.11
uDeepGRU$_3$	0.46	-1.87
3-cent	1.61	-0.70
DeA	1.01	-1.31
AJ-RN	0.67	-1.65
PI-RN	-0.67	-2.99
Segment LSTM$_1$	5.56	3.23
Segment LSTM$_2$	1.91	-0.41

In Table 5.1, we observe that all uDeepGRU variations achieved the highest recognition accuracy as well as the lowest mislabeled classifications. These variations did, however, demonstrate some false as well as missed detections. The variation which utilized an ensemble of networks produced the best overall result.

Looking at the latency values (Table 5.2), we observe that most methods were quick to spot the gesture starts and output a recognition before the gesture ends. PI-RN demonstrated negative lag,

and was able to recognize gestures even before they started. Again the ensemble variation of uDeepGRU demonstrated the best overall performance.

5.5 Improving uDeepGRU

After participating in the SHREC 2019 challenge, we incorporated additional improvements to the uDeepGRU model. Our efforts were guided by experimentation across additional datasets, along with the lessons we learned while working towards making a submission to the SHREC 2019 challenge. We provide an overview of our efforts and present an evaluation of our final model.

5.5.1 Architectural Tweaks

We introduced small changes to the original uDeepGRU architecture. We removed all batch normalization and dropout layers, and used a simple FC layer with 512 hidden units and $tanh()$ activation as the first layer of our network before the first GRU layer. This addition was motivated by the idea of *feature embedding* which is typically done in language models [155]. We found this addition helpful in increasing the model's accuracy as it acts as an initial feature extractor for the network. We additionally considered feature embedding via convolution layers, another common practice in time-series analysis and also experimented with alternate activation functions such as ReLU or sigmoid. We did not observe any noticeable benefits with these changes.

5.5.2 Revisiting the Training Loss Function

We discussed the original version of uDeeGRU's loss function in Section 5.3.1. This training objective consisted of minimizing the cross-entropy loss as well as the negative Sorensen-Dice

coefficient, which in turn maximized the F_1 score between the ground truth and the predicted labels. We investigated several ideas before deciding on the final training loss function to use for the improved uDeepGRU model.

As mentioned in Section 3.2.1, the use of probabilistic models such as conditional random fields (CRF) for sequence labeling has been extensively explored in the literature. Inspired by the recent success of such models in natural language processing tasks [156], we experimented with learning the transition scores of a linear-chain CRF model as a part of uDeepGRU training. The resulting model performed well, yet its accuracy did not exceed that of the original uDeepGRU model.

Next we investigated the use of sDTW scores [46] as a measure of the dissimilarity between the ground truth labels and the predicted labels. In simple terms, given an input example $\mathbf{x}$ the training process in this case optimizes uDeepGRU's parameters such the sDTW dissimilarity between the predicted output labels $\hat{\mathbf{y}} = \{\hat{y}_0, \hat{y}_1, ..., \hat{y}_n\}$ and the ground truth labels $\mathbf{y} = \{y_0, y_1, ..., y_n\}$ is reduced. Formally, this is equivalent to minimizing:

$$\mathscr{L} = \text{sDTW}\big(\mathbf{y}, \hat{\mathbf{y}}; f\big) \tag{5.2}$$

where f is the internal cost function for the sDTW algorithm. We experimented with the Euclidean distance as well as cosine dissimilarity for this internal cost function. Our early experiments with this formulation in fact showed promising results, where the F_1 score increased by as much as 1%. However, training times became longer as the sDTW algorithm is inherently more computationally intensive than our original loss formulation.

Our last final attempt to formulate a better loss function to train uDeepGRU with was guided by manual error analysis on the model's outputs. We observed that the majority of the misclassified labels were due to missed detections: uDeepGRU would conservatively label many frames as "no

gesture". This stems from the unbalanced nature of the continuous gesture recognition problem. In most such problems, a majority of user interactions consist of non-gestural interactions while the actual gestures occupy a smaller portion of the input data. Thus a recognizer trained on such an unbalanced dataset often leans towards outputting the label for the majority class, which in this case is the "no gesture" class.

Although the use of the Sorensen-Dice coefficients in our original loss formulation was to mitigate such effects, other formulations have been studied in the literature. Notably, Lin *et al*. recently introduced *focal loss* [157] which adds a dynamic weighting factor to the standard cross-entropy. Assuming that the network outputs the probability p for a given class label the focal loss is defined as:

$$\mathscr{L}_{\text{focal}} = -(1-p)^{\gamma} \log(p) \tag{5.3}$$

where $\gamma \geq 0$ is the *focusing* parameter that determines how much emphasis is put on misclassified examples ($\gamma = 0$ yields the original cross-entropy loss value). In practice, focal loss worked remarkably well for us leading to increases of up to 5% in our F_1 scores when $\gamma = 1$. Additionally, this loss function is fairly straightforward to implement and significantly reduces the complexity of the optimization problem. Taking all aspects into consideration, we decided to train our improved uDeepGRU model via minimizing Equation 5.3 alone.

5.6 Evaluating the Improved uDeepGRU Model

We designed a set of experiments to evaluate the improved uDeepGRU model. Our main evaluation metrics consist of accuracy, F_1 score, precision and recall which are not only widely adopted measures but are also suitable for the unsegmented gesture recognition problem due to class imbalance.

Datasets. We selected four different datasets for our evaluations, spanning across different gesture modalities and input devices: Montalbano v2 [2] (14000 samples of 20 Italian sign language utterances, collected from Kinect v1), the unsegmented version of DHG 14/28 [4] called *Online DHG*[1] (280 samples of 10 hand gestures collected from a near-view Intel RealSense depth), UT-Kinect [3] and SHREC 2019 [1]. All of these datasets contain segmented training sequences and unsegmented testing sequences. Note that although UT-Kinect [3] was not specifically designed for continuous recognition, we found sufficient frame-level labels to adopt it for our evaluations. Across all datasets we performed the same set of data augmentation techniques as our SHREC 2019 submission, except for sequence inversion which we found to hurt the performance in some cases. Lastly, some of these datasets contained data from other gesture modalities (such as videos), yet we only used the skeleton data for our tests.

Recognizers. We compare uDeepGRU against various recognizers: support vector machine (SVM), random forest, naïve Bayes, decision trees, AdaBoost and DeepGRU. These recognizers represent a variety of commonly used methods in the literature of gesture recognition. Except for DeepGRU, all other methods require explicit feature extraction for which we use the commonly used Rubine [48] feature set extended to 3D gestures [158]. Since none of these methods were specifically designed for continuous recognition, we implemented a sliding window recognizer based on each one. The length of this window was set to 20 frames established as via cross-validation.

5.6.1 Discussion

Tables 5.3 to 5.6 summarize the results of our experiments. All evaluations were performed on raw outputs without any post-processing on the output labels. The original and the improved uDeepGRU models are signified as v1 and v2 respectively. The results in each table are sorted in

[1]Dataset available at: *http://www-rech.telecom-lille.fr/shrec2017-hand/*

Table 5.3: Recognition results on SHREC 2019 [1] (all values are percentages).

Recognizer	**F1**	**Accuracy**	**Precision**	**Recall**
Naïve Bayes	28.2	58.2	22.8	36.8
Decision Trees	35.6	80.8	32.2	39.7
AdaBoost	37.4	86.5	38.6	36.3
Random Forest	51.6	90.4	61.0	44.7
SVM	60.4	91.8	69.3	53.5
DeepGRU	66.7	91.3	65.4	68.0
uDeepGRU$_{v1}$	68.3	92.3	68.6	68.1
uDeepGRU$_{v2}$	**73.6**	**94.0**	**77.8**	**69.9**

Table 5.4: Recognition results on Montalbano v2 [2] (all values are percentages).

Recognizer	**F1**	**Accuracy**	**Precision**	**Recall**
Naïve Bayes	18.5	28.4	16.7	20.8
AdaBoost	25.1	60.3	48.1	17.0
Decision Trees	31.7	56.7	31.2	32.3
Random Forest	43.8	68.1	55.0	36.4
SVM	56.9	74.1	68.9	48.4
DeepGRU	66.8	79.8	71.8	62.5
uDeepGRU$_{v1}$	69.7	81.5	73.8	66.0
uDeepGRU$_{v2}$	**70.7**	**82.1**	**75.2**	**66.7**

the increasing order of F_1 score. Across all experiments, we observe that the top three performing recognizers are the models that we proposed and discussed in this work. We also observe a large performance gap between these models and the classical machine learning methods such as SVM and random forest, despite the popularity of the latter models in various gesture recognition tasks.

Focusing on the DeepGRU family of recognizers, we note that the improved uDeepGRU model outperforms its original counterpart which highlights the importance of the additional changes that we included in the improved version. The benefit of using focal loss for training is also evident in these results. This improved model additionally outperforms the windowed DeepGRU recognizer.

Table 5.5: Recognition results on UT Kinect [3] (all values are percentages).

Recognizer	F1	Accuracy	Precision	Recall
AdaBoost	32.6	36.4	34.6	30.8
Decision Trees	33.3	52.3	35.6	31.2
Naïve Bayes	37.0	27.0	28.6	52.5
Random Forest	37.6	62.8	53.7	28.9
SVM	48.2	68.9	54.1	43.4
DeepGRU	52.1	65.5	55.0	49.5
uDeepGRU$_{v1}$	61.2	65.9	61.8	60.6
uDeepGRU$_{v2}$	**66.9**	**70.9**	**64.1**	**70.0**

Table 5.6: Recognition results on Online DHG [4] (all values are percentages).

Recognizer	F1	Accuracy	Precision	Recall
Naïve Bayes	23.7	23.7	18.3	33.6
Decision Trees	28.5	54.4	24.9	33.3
AdaBoost	37.3	69.8	37.1	37.6
Random Forest	38.2	70.0	37.4	39.1
SVM	45.5	71.7	45.0	46.0
DeepGRU	53.3	75.0	53.0	53.6
uDeepGRU$_{v1}$	55.3	76.3	54.4	56.2
uDeepGRU$_{v2}$	**58.6**	**77.7**	**58.0**	**59.1**

Considering DeepGRU's segmented recognition performance on UT-Kinect (Table 4.1), these results are remarkable as DeepGRU previously achieved the recognition accuracy of 100%. Yet, training the same model on the continuous version of this dataset yields mediocre results, which highlights the challenging nature of continuous gesture recognition.

In closing, we note that from a practical point of view, F_1 score applied to frame-level labeling may be too strict to measure the performance of a continuous recognizer from the usability point of view. The users' interaction experience with a gesture-based interface ultimately hinges on how well the system can detect an intended action, rather than how many times during the performance

of the action the system was able to output correct detections. In other words, a user who intends to execute a punching motion expects the system to register the action as such once the motion is completed. Despite these, the F_1 score is a widely adopted measure in the recognition literature, which is why we adopted it for this work.

5.7 Conclusion

In this chapter, we discussed the architecture of uDeepGRU, a deep-learning based network for online gesture recognition. We discussed how we adapted DeepGRU to online gesture recognition tasks. We discussed the original uDeepGRU model and the results of SHREC 2019 Online Gesture Recognition challenge which showed the effectiveness of our proposed method. We then turned our focus to the improvements that we made to the network's architecture since the original submission. These improvements consisted of adding an embedding layer at the beginning of the network and adopting focal loss instead of minimizing cross-entropy and negative Sorensen-Dice score. We also discussed some other improvement ideas which we explored but found unhelpful.

We evaluated both versions of uDeepGRU across four different datasets and compared their performance against other recognizers. In all experiments, the improved uDeepGRU model outperformed all other recognizers both in accuracy and F_1 score.

CHAPTER 6: GESTURE GENERATION WITH RECURRENT NEURAL NETWORKS

6.1 Introduction

In this chapter we turn our focus to gesture generation using deep recurrent neural networks. As stated previously, GANs have been explored extensively for image generation tasks, but the generation of real-valued sequences, and especially gestures has not received much attention in the literature. We focus on *modality-agnostic* gesture generation wherein gestures are represented by a sequence of 2D or 3D positional features typically produced by touch interfaces, Kinect or similar input devices. The overarching goal is to address the challenges involved with training GANs, such as challenge the need for training two networks (the generator and the discriminator) simultaneously which can lead to long training times.

We start by discussing *DeepGAN*, our novel GAN approach for dynamic gesture generation, through which our deep recurrent gesture generator network is born. We thereafter discuss our unique solution for alleviating the difficulties associated with training GANs. Specifically, we formulate a novel and intuitive loss function for training our gesture generator. Our loss function, which is based on the dynamic time warping (DTW) algorithm [42], completely replaces DeepGAN's discriminator network. This, transforms the significantly complex adversarial training procedure of a GAN to the much simpler *non-adversarial* training problem: train the generator by minimizing a loss function that directly maps the quality of the generated samples to their similarity to the real ones. We call this approach *DeepNAG*. We evaluate both methods by using their generated gestures in data augmentation for improved gesture recognition across a variety of datasets of different sizes and modalities, as well as different gesture recognizers.

6.1.1 Task Definition and Goals

Given a dataset $\mathbb{D}$ of gesture samples, our task is to learn the distribution of the samples through training a generator network G. Because our underlying data are time-series, the natural choice for G is an RNN, as explained in Section 2.4.1. We leverage GAN models (see Section 2.4.1), as well as a novel formulation of a generator network to model the distribution of $\mathbb{D}$.

In addition to generating synthetic samples, we show that the generalization ability of DeepGRU is not limited to recognition tasks. Rather, we reuse many components of DeepGRU in designing DeepGAN and DeepNAG.

6.2 Gesture Generation with GANs

Our initial approach for gesture generation uses the well-known GAN training setting comprised of a generator and a discriminator. We call this recurrent model *DeepGAN*, which we designed incrementally and was informed by the latest developments in deep learning. Early on, the simplicity and the recognition power of the DeepGRU model inspired us to adopt it as our discriminator. Through experiments guided by our ablation on DeepGRU (see Section 4.4), and with the goal of managing design complexity, we settled for the simpler uDeepGRU variant as our discriminator.

One typical problem encountered in training GANs is that during training, one network may overpower the other. This can cause various problems, including non-convergence or increased training times. As such, when selecting the neural architecture for either D or G, a typical safe choice is to select similar architectures for both D and G and equalize the number of training parameters. Although there is no consensus or guideline about the choice of architectures, our experience has shown that using similar architectures for both networks reduces the complexity and the uncertainty of the training process. We conducted experiments across different datasets with generators con-

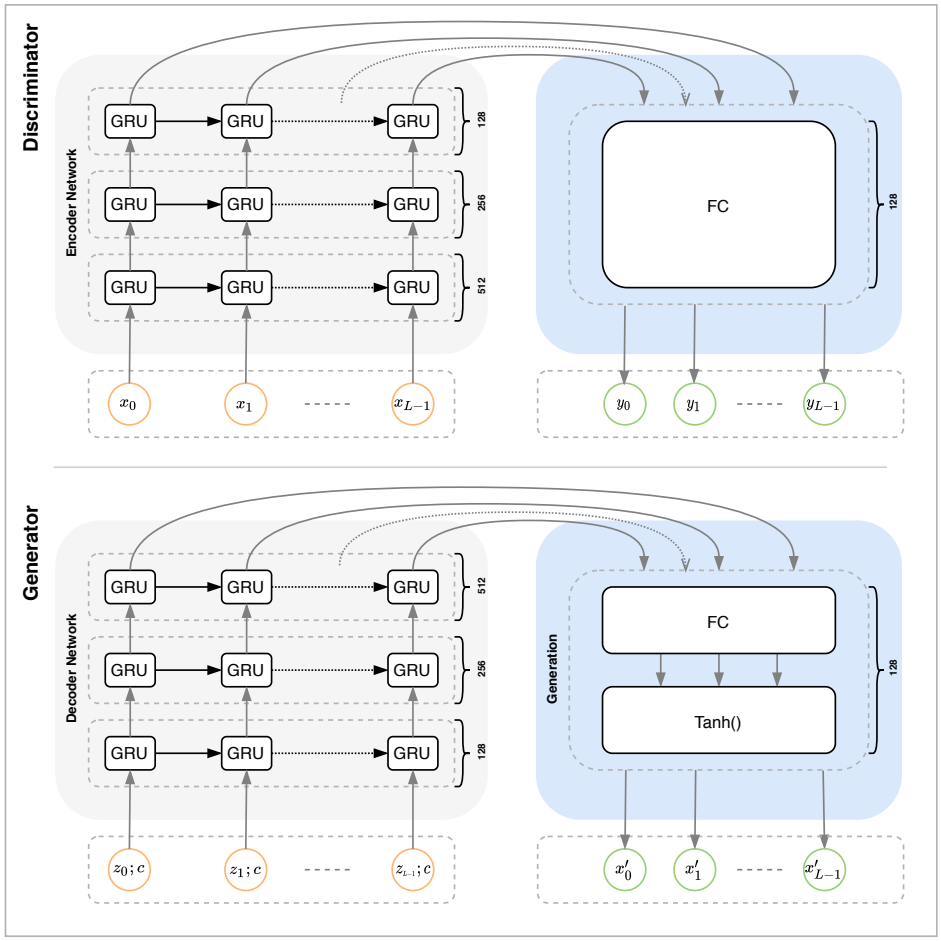


Figure 6.1: **DeepGAN** – our proposed model for gesture generation. DeepGAN consists of the discriminator and generator networks.

sisting of both LSTM and GRU units, as well as a varying number of recurrent layers. We observed more stable training, less overfitting and more plausible outputs with a decoder-style network resembling the *flipped* version of our discriminator. A possible explanation for this could be that this choice potentially benefits from the balance between the two D and G networks. Figure 6.1 depicts the architecture of DeepGAN, which we believe is easy to understand and straightforward to implement in any modern deep learning framework. A common design for generators is the use of the $tanh()$ activation function in the last layer, which is what we use as well.

To generate a gesture sample $\mathbf{x}'$ of class c, a *class-conditioned* latent vector $\mathbf{z}_{\hat{c}}$ is created and fed to G to create a sample $\mathbf{x}'$ as follows:

$$
\begin{aligned}
\mathbf{z} &= \Big\{ z_i;\ \ \forall z_i \sim \mathcal{N}\big(0,1\big) \Big\} \\
\hat{\mathbf{c}} &= \text{one-hot}\big(c\big) \\
\mathbf{z}_{\hat{c}} &= \Big\{ [\ z_i; \hat{\mathbf{c}}\ ],\ \ \forall z_i \in \mathbf{z} \Big\} \\
\mathbf{x}' &= G_\theta\big(\mathbf{z}_{\hat{c}}\big)
\end{aligned}
\tag{6.1}
$$

In simple terms, $\mathbf{z}$ contains per time step random noise vectors. Thus, given that we intend to generate sequences with a fixed length of 64 time steps, $\mathbf{z}$ contains 64 elements. Each time step $z_i \in \mathbf{z}$ is sampled independently from the standard normal distribution, and class-conditioning is done by appending the *one-hot* representation of c to each time step which avoids ignoring the conditioning through forgetting [24]. Also, we fixed the dimensionality of the latent space to 32.

6.2.1 Loss Function

We experimented with different loss functions to train DeepGAN. Even though training with the classic adversarial loss [21] yielded plausible results, we observed improved sample quality and better convergence with the improved Wasserstein loss [28], which is what we use in this work. Specifically, we minimize with gradient descent $\mathscr{L}_D$ and $\mathscr{L}_G$ as defined below:

$$\hat{\mathbf{x}} = \epsilon \mathbf{x} + (1 - \epsilon)\mathbf{x}' \tag{6.2}$$

$$\mathscr{L}_D = D_\theta\big(\mathbf{x}'\big) - D_\theta\big(\mathbf{x}\big) + \lambda\Bigg(\Big\|\nabla_{\hat{\mathbf{x}}} D_\theta(\hat{\mathbf{x}})\Big\|_2 - 1\Bigg)^2$$

$$\mathscr{L}_G = -D_\theta\Big(G_\theta\big(\mathbf{z}\big)\Big)$$

where $\epsilon \in \mathcal{U}(0, 1)$ and λ controls the gradient penalty as described in Section 2.4.1.2.

6.2.2 Training Procedure

We train DeepGAN end-to-end: we sample mini-batches of size 64 from the training data, and optimize both D and G networks at the same time. As a pre-processing step, we scale all available training data to the range $[-1, 1]$. Also, to reduce the complexity of working with gestures of different sizes, we resample all gesture sequences in $\mathbb{D}$ to the length $L = 64$, thus both the inputs as well as the outputs of DeepGAN have the same length.

We employ the same optimization procedure and the hyperparameter set as [28]. Specifically, we use the Adam solver [7] ($\beta_1 = 0, \beta_2 = 0.9$) with a learning rate of 10^{-4}. We set the gradient penalty coefficient $\lambda = 10$ and train D five times more frequently than G. Given the importance of the gradient penalty coefficient λ, we performed a set of experiments to examine the effects of the choice of this parameter. Figure 6.2 summarizes these effects on our training plots. Smaller values of this coefficient led to unstable training, while larger values prevented convergence. Figure 6.3 depicts a small number of generated samples when DeepGAN is trained with different values of λ. Note that some generated samples are rather noisy. Although both $\lambda = 5$ and $\lambda = 10$ were suitable candidates, we settled for the latter which is what Gulrajani *et al*. [28] used.

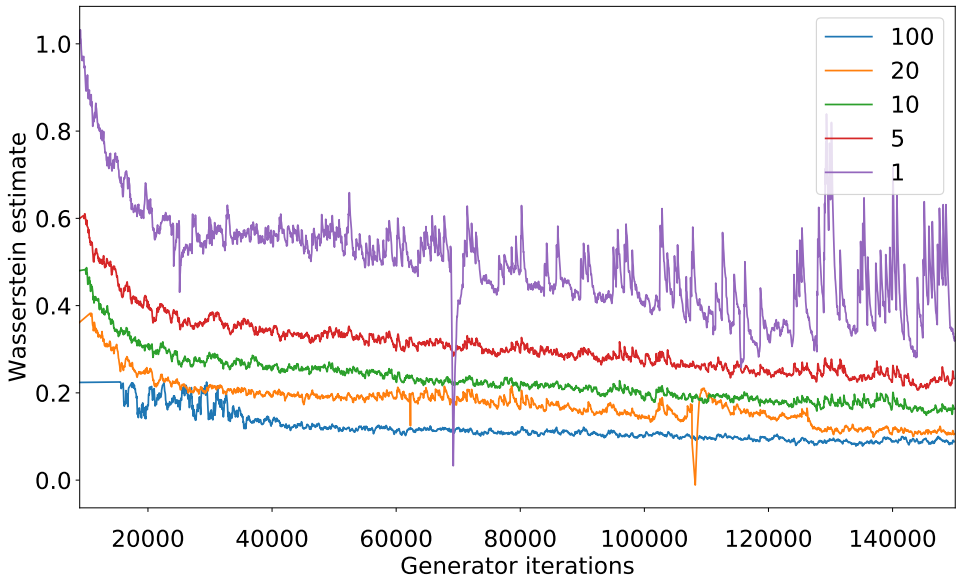
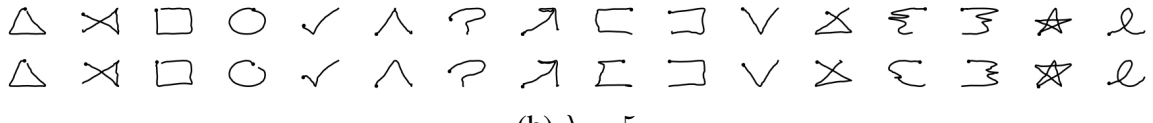
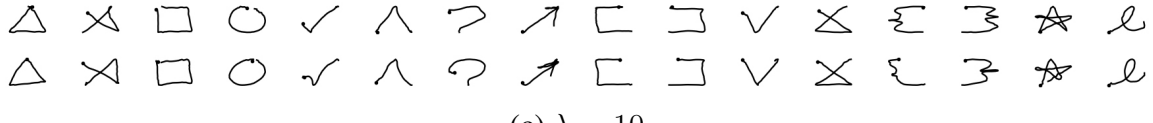


Figure 6.2: Training curves of DeepGAN on $1-GDS [62] using different values of λ.

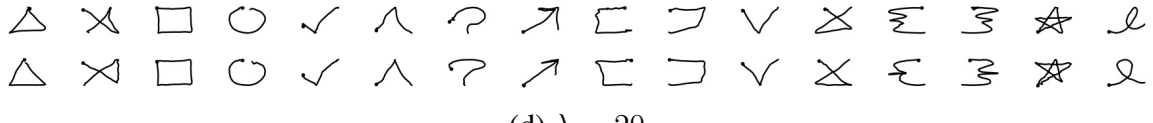
(a) $\lambda = 1$



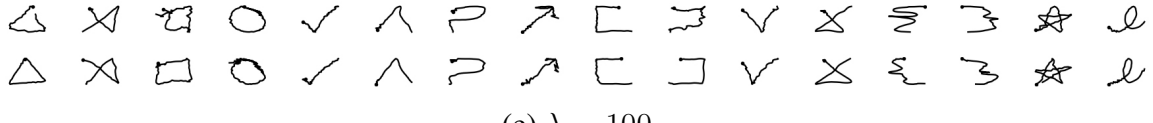
(b) $\lambda = 5$



(c) $\lambda = 10$



(d) $\lambda = 20$



(e) $\lambda = 100$

Figure 6.3: Generation results after training using various values of λ. Note the noise in some of the generated samples.

6.3 Towards Non-Adversarial Gesture Generation

While DeepGAN shows promising results (see Section 6.5), it demonstrated a few shortcomings early on. Most importantly, generating synthetic samples requires the training of two networks at the same time, which increases the burden on the developer: every decision about the network design has to have the balance between the generator and the discriminator in mind, all hyperparameters need to be tuned twice, and changes to one model may have adverse effects on the other. Moreover, the training times were long and typically exceeded 72 hours. These issues contradict our *accessibility* philosophy and design goals. With these challenges and our design goals in mind, we aim to simplify the process of generating synthetic gestures with deep networks.

Perhaps the key to simplifying the network design and reducing the burden of working with GANs is the answer to a fundamental question: do we really need two separate networks? Is it possible to train a single network that is capable of generating synthetic samples? To answer this question, we turn our focus to the reason GANs typically contain two separate networks.

As stated previously, a generator network aims to learn the distribution of the underlying dataset $\mathbb{D}$, so that new samples can be sampled from the distribution. It typically does so by learning how to fool a discriminator, whose aim is to distinguish between real and fake instances. In Wasserstein GANs, the discriminator is replaced by a critic, whose goal is to judge how well fake samples correlate with the real ones by learning the non-linear distribution of real samples. These designs, which involve a discriminator or a critic, are necessary because a measure for distribution similarity, or how similar fake samples to the real ones may not be readily available, or easily derivable. Also, the Wasserstein distance formula is intractable [27], which is why it is replaced by a critic network.

While GANs are intended to serve as a general framework for learning sample distributions and generating synthetic samples across a variety of domains, our goal, which is generating synthetic

gestures, is rather well-defined and constrained: generate synthetic gestures that are *similar* to their real counterparts. The fundamental question then becomes, how would one define the similarity of two gestures, and how to train a generator that increases the similarity between fake and real gestures? The answer is surprisingly simple: by reducing the *dissimilarity* between the two! Thinking of gestures as temporal sequences, the very first dissimilarity measure that comes to mind is dynamic time warping [42]. Armed with this intuition we stride towards formulating our novel loss function for gesture generation.

6.4 DeepNAG

In what follows, we show how to formulate the problem of synthetic gesture generation as the training of a single recurrent network whose goal is to minimize a single loss function rather than fooling a discriminator. We call our synthetic gesture generator *DeepNAG*: a non-adversarial gesture generator. Since DeepGAN showed promise in generating synthetic samples, we reuse its generator for DeepNAG (see Figure 6.1).

Our task remains the same as what we described in Section 6.1.1: given the class-conditional latent variable $\mathbf{z}_{\hat{c}}$, generate synthetic samples $\mathbf{x}' = G_\theta\big(\mathbf{z}_{\hat{c}}\big)$. However, we are interested in learning the trainable parameters θ via minimizing a single loss function that directly provides a measure for the similarity between real samples $\mathbf{x}$ and the generated ones $\mathbf{x}'$. Obviously, the loss function should be differentiable to be usable in deep network training.

As mentioned in Chapter 3, DTW [42] is a popular algorithm for measuring the dissimilarity between time-series The vanilla DTW formulation is not differentiable, because the shape of the DTW function is non-smooth. Recently, Cuturi and Blondel [46] introduced a differentiable formulation of DTW, called soft DTW (sDTW) which can be used for gradient-based optimization

tasks. We leverage this formulation for sequence generation. Considering our goal with DeepNAG, which is to increase the similarity of a real sample $\mathbf{x}$ and a fake one $\mathbf{x}' = G_\theta\big(\mathbf{z}_{\hat{c}}\big)$, one could naïvely decide to learn θ by minimizing:

$$\mathscr{L} = \text{sDTW}\big(G_\theta(\mathbf{z}_{\hat{c}}), \mathbf{x}; f\big) \tag{6.3}$$

where f is the cost function of Equation 2.12, typically set to either the Euclidean distance (ED) or the cosine similarity (COS). Unfortunately, this naïve formulation merely states that generated samples $\mathbf{x}'$ should be as similar to the real samples $\mathbf{x}$ as possible, without regards to the variations of the real samples. Observant readers immediately realize that a trivial solution to this optimization problem is the mean (centroid) sample $\bar{\mathbf{x}} = \frac{1}{n} \sum_{\mathbf{x} \in \mathbb{D}} \mathbf{x}$. In other words, $\bar{\mathbf{x}}$ is the one possible value that is the most similar to *all samples* in the dataset.

To avoid the pitfall of a trivial solution and account for inter-class variations, we ask ourselves what defines the *realism* of fake samples? An obvious answer would be, how reminiscent of the real samples they are. In other words, how closely they mimic the features, variations and subtleties of the real ones. This naturally means the distribution of the synthetic samples should sufficiently *cover* the distribution of the real ones.

With this in mind, we use the Hausdorff distance [159], which is a well-studied measure of point-set similarity that is easy to compute and implement. Hausdorff distance computes a distance between the two sets by taking into account the data points in each dataset that have the extreme (minimum/maximum) distances from one another. Conveniently, the average Hausdorff distance (denoted as $d_{\mathcal{H}}$ henceforth) between point sets $\mathbf{A}$ and $\mathbf{B}$ is differentiable [160] and is defined as:

$$d^f_{\mathcal{H}}(\mathbf{A}, \mathbf{B}) \;=\; \frac{1}{|\mathbf{A}|}\sum_{a\in\mathbf{A}} \min_{b\in\mathbf{B}}\; d\Big(a, b; f\Big) \;+\; \frac{1}{|\mathbf{B}|}\sum_{b\in\mathbf{B}} \min_{a\in\mathbf{A}}\; d\Big(b, a; f\Big) \tag{6.4}$$

where $d\big(a, b; f\big)$ is the distance (dissimilarity) between two points a and b parameterized by f. We use $d\big(a, b; f\big) = \text{sDTW}(a, b; f)$, and will discuss the choice of f (sDTW's cost function) shortly. Finally, we propose the following as DeepNAG's loss function to minimize. To our knowledge, this is the very first formulation of a single loss measure to train a deep recurrent neural network for generating synthetic gesture sequences:

$$\mathscr{L}_f(\mathbf{X}', \mathbf{X}) = \underbrace{d^f_{\mathcal{H}}\big(\mathbf{x}'_1, \mathbf{x}_1\big)}_{\text{Similarity term}} \;+\; \underbrace{\Big|d^f_{\mathcal{H}}\big(\mathbf{x}'_1, \mathbf{x}'_2\big) - d^f_{\mathcal{H}}\big(\mathbf{x}_1, \mathbf{x}_2\big)\Big|}_{\text{Variation term}} \tag{6.5}$$

where $\mathbf{X}' = \{\mathbf{x}'_1, \mathbf{x}'_2\}$ (two generated samples), and $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2\}$ (two real samples), and both sample sets belong to the same gesture class. We only use the derivative $\partial\mathscr{L}_f/\partial\mathbf{x}'_1$ during training since it yielded good results and was faster. Intuitively, Equation 6.5 expresses that training G should aim to increase the similarity of fake and real samples[1] (similarity term), while maintaining the similarity balance between two batches of fake and real samples (variation term). The former term ensures real and generated samples are similar, while the latter term ensures generated samples maintain proper overall inter-class variations, effectively avoiding pitfalls such as mode-collapse typically encountered in GANs.

[1] In other words *decrease* their *dissimilarity*.

6.4.1 Practical Notes

When computing sDTW(a, b) we ensure *class-awareness*: the value is only computed if samples a and b belong to the same gesture class. As for the choice of sDTW's internal cost function f, we started with ED, but the benefits of using COS quickly became apparent to us: minimizing $\mathscr{L}_{\text{ED}}$ yielded high-quality results but convergence was slow. Conversely, minimizing $\mathscr{L}_{\text{COS}}$ led to much faster convergence with sometimes noisier results. In the end, we settled for minimizing both and leave a thorough study on the effects of each cost function to future work. Lastly, recall our use of fixed-length sequences ($L = 64$) where the points in the sequence are equidistant. To enforce the production of such sequences by G we add an additional term $\mathscr{L}_{\text{Resample}}$ to our objective. Putting everything together, the following is the loss function that we minimize for our experiments:

$$\mathscr{L}_{\text{DeepNAG}}\big(\mathbf{X}', \mathbf{X}\big) = \mathscr{L}_{\text{ED}}\big(\mathbf{X}', \mathbf{X}\big) + \mathscr{L}_{\text{COS}}\big(\mathbf{X}', \mathbf{X}\big) + \alpha \cdot \mathscr{L}_{\text{Resample}}\big(\mathbf{x}'_1\big) \tag{6.6}$$

$$\mathscr{L}_{\text{Resample}}\big(\mathbf{x}'\big) = \frac{1}{|\vec{\mathbf{x}'}|} \sum_{\forall \vec{x'_i} \in \vec{\mathbf{x}'}} \left(\left\| \vec{x'_i} \right\| - \widetilde{L}(\mathbf{x}') \right)^2 \quad , \qquad \widetilde{L}(\mathbf{x}) = \frac{1}{|\vec{\mathbf{x}}|} \sum_{\forall \vec{x_i} \in \vec{\mathbf{x}}} \left\| \vec{x_i} \right\|$$

where $\widetilde{L}(\mathbf{x})$ is the length of each $\vec{x_i} \in \vec{\mathbf{x}}$ after $\mathbf{x}$ is resampled to L equidistant points. Thus, $\mathscr{L}_{\text{Resample}}$ simply enforces that points in $\mathbf{x}'$ be equidistant with α as its regularizer. Note that minimizing $\mathscr{L}_{\text{ED}}$ alone (Equation 6.5) yields good-quality results in most cases. However, we achieved faster convergence and better data augmentation performance using Equation 6.6.

6.4.2 Training Procedure

We train DeepNAG end-to-end: we sample mini-batches of size 64 from the training data, and optimize DeepNAG using the Adam solver [7] with the same hyperparameters as DeepGAN, and employ the same scaling and resampling procedure to train DeepNAG. Lastly, we used $\gamma = 0.1$ and $\alpha = 10^3$. All hyperparameters were established experimentally and via cross-validation.

Synthetic production results on the $1-GDS [62] using both DeepGAN and DeepNAG are depicted in Figure 6.4. At a glance, both networks produce visually compelling results and samples demonstrate diversity.

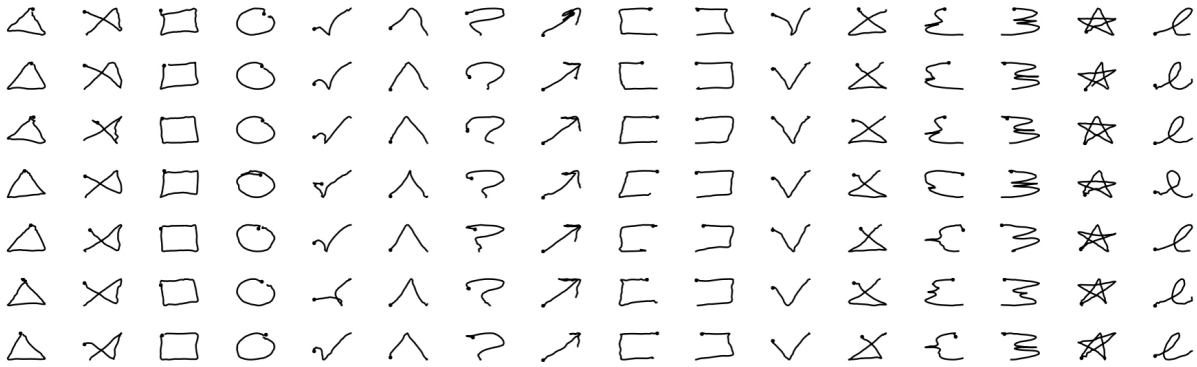
6.4.3 Implementation

We implemented DeepGAN and DeepNAG with the PyTorch [11] framework. Our reference implementation is accessible at *https://www.deepnag.com*. Additionally, our implementation requirements yielded multiple other standalone projects, which we have publicly released in the hope of benefiting the deep learning community.

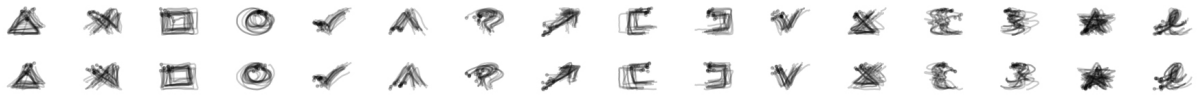
Inspired by [161], we implemented a fast version of sDTW in CUDA with a PyTorch interface using Numba [162]. Our novel implementation parallelizes both forward and backward passes, and runs up to $100\times$ faster than any other publicly available implementation that we know of. Our public repository[2] has gained traction among the practitioners since its release. We also implemented fast GRU units using PyTorch's just-in-time (JIT) compilation features to allow computing their higher-order derivatives, a feature that is missing in PyTorch[3]. Such derivatives are required

[2]Fast sDTW for PyTorch: *https://github.com/Maghoumi/pytorch-softdtw-cuda*

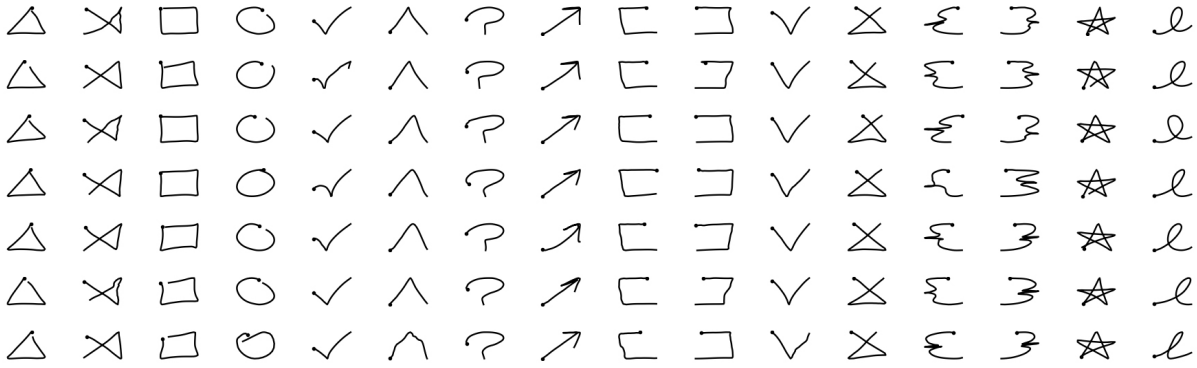
[3]To our knowledge, the cuDNN framework is missing this feature. Thus, at the time of this writing one cannot compute higher-order derivatives for GPU-based GRUs in any deep learning framework that relies on cuDNN.



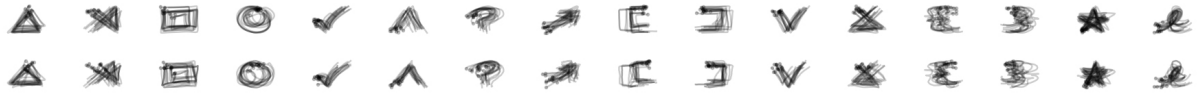
(a) DeepGAN (samples)



(b) DeepGAN (overlays)



(c) DeepNAG (samples)



(d) DeepNAG (overlays)

Figure 6.4: Some synthetic gestures produced by DeepGAN (a) and DeepNAG (b) when trained on $1-GDS [62]. Overlayed rendering of synthetic samples (top row in each group), and real samples (bottom row in each group) are shown. Each overlay consists of 16 samples per class. For both DeepGAN and DeepNAG, the diversity of the synthetic samples resemble that of the real ones.

to implement the improved WGAN loss [28] for GRUs. Our repository is publicly available[4].

6.5 Evaluation

In this section, we evaluate DeepGAN and DeepNAG from two practical aspects, namely *quantitative* and *qualitative*. Our quantitative evaluation consists of using each model for generating synthetic data for improved gesture recognition (*i.e.* data augmentation). This evaluation determines whether our generators are appropriate for improving gesture recognition accuracy. Our qualitative evaluation consists of a user study on Amazon Mechanical Turk. This evaluation provides insight into the visual quality of the generated samples and gauges the human's perception of the realism of our synthesized samples.

A demo video showing generated samples from DeepNAG is available at *https://www.maghoumi.com/dissertation*.

6.5.1 Quantitative Evaluation

We quantitatively evaluate DeepGAN and DeepNAG in data augmentation tasks focusing on scenarios with limited training data. Given a dataset of gestures collected from multiple participants, we simulate small training sets by splitting the data into training (50%), validation (20%) and test (30%) sets. Our experiments are all *subject-independent* [45]; *i.e.* the data of each participant only appears in one of these sets. This is a more challenging and realistic evaluation protocol, as it ensures that during training, the recognizer never sees any data from the participant it will be evaluated on during the validation and testing phases.

[4] Just-in-time GRUs for PyTorch: *https://github.com/Maghoumi/JitGRU*

We begin by training a gesture recognizer on the training set, and use the validation set for model selection. We evaluate the best performing model on the test set and record its recognition error (baseline). Next, we augment the training set with a selected data generation method and repeat the experiment: train the recognizer with this new training set, use the validation data for model selection, and evaluate the best model on the test set. We record the recognizer's recognition error again, which will be the error after augmenting the training set. Comparing this number with the baseline benchmarks the synthetic data generation method. In total we perform 150 experiments: we train five gesture recognizers on six different datasets to evaluate four synthetic data generation methods against the baseline.

Datasets. We selected six datasets among the ones frequently studied in the literature. They vary in size and span across gesture modalities and input devices: JK2017 (Kinect) [44] (14 full-body fighting gestures of 20 participants with Kinect v2), JK2017 (Leap Motion) [44] (eight hand-gestures of 20 participants with Leap Motion), UT-Kinect [3] (ten full-body daily activities of ten participants with Kinect v1), MSR Action 3D [163] (20 full-body actions of ten participants with Kinect v1), SBU Kinect Interactions [129] (8 two-person interaction of seven participants with Kinect v1) and $1-GDS [62] (16 2D pen gestures of ten participants).

Recognizers. We selected five gesture recognizers: support vector machine (SVM), random forest, naïve Bayes, DeepGRU and Jackknife [44]. These represent classic machine learning algorithms, deep learning as well as rapid prototyping [44] approaches, which are common choices for gesture recognizers. The first three methods require explicit feature extraction for which we use the commonly used Rubine [48] feature set extended to 3D gestures [158]. Jackknife [44] is a 1-nearest neighbor DTW-based template matching recognizer.

Data generation methods. We compare four data generation methods against the baseline: random Gaussian noises, GPSR [108], DeepGAN and DeepNAG. Note that we found GPSR to be ben-

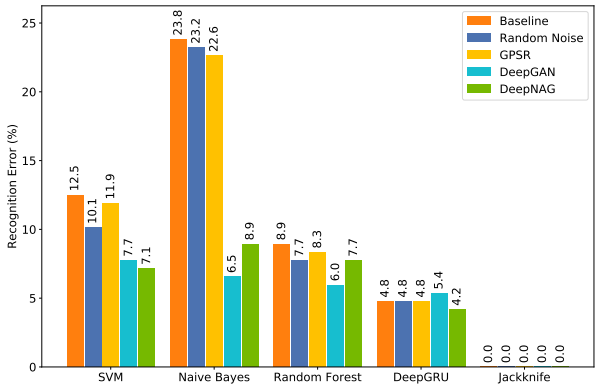
eficial for DeepGRU and uDeepGRU evaluations, which is why we decided to compare against it.

Hyperparameters. All hyperparameters were tuned across different datasets, but the same set of parameters were used for every experiment. Both DeepGAN and DeepNAG were trained on the 50% split training set, and shared most hyperparameter settings as described previously. Other parameters were chosen via cross-validation as follows. GPSR parameters were set to $r = 2, \sigma = 0.25$ and the magnitude of random noise was set to 2% of the bounding box of each feature.

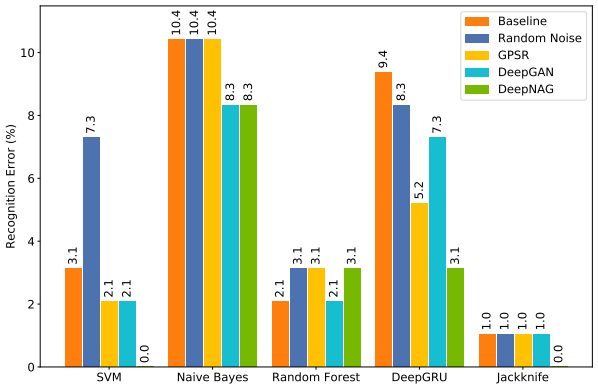
6.5.1.1 Results and Discussion

Figure 6.5 depicts the results of our experiments. In many cases the use of some form of data augmentation decreases the recognition error, indicating that our 50% split to simulate small training sets is working as expected. To better contrast the generation methods we employ a scoring scheme that quantifies whether the use of a given augmentation method is both *warranted* and *effective*. Data augmentation is only warranted if a recognizer trained with the additional data outperforms the baseline. Additionally, a method is effective only if it outperforms random noise. We start with a score of zero for a given generator. In each experiment set, we increment this score if the method outperforms all other methods in addition to the baseline and random noise. Ties are only counted if the method outperforms both random noise and the baseline, and we use the cumulative score for comparison.

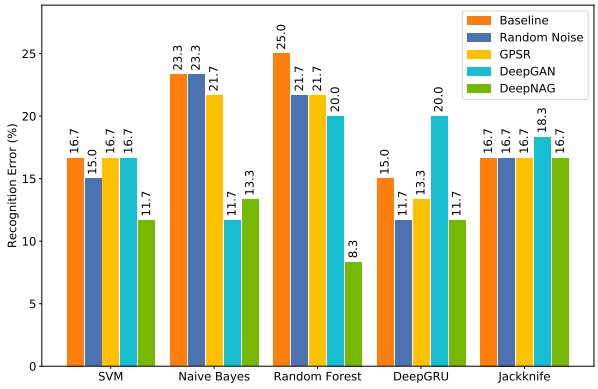
Table 6.1 presents the computed score aggregates over each dataset and recognizer. We observe that across both aggregate groups, DeepNAG outperforms other methods by a large margin, suggesting its suitability for data augmentation regardless of the choice of dataset or recognizer. In a few cases, DeepNAG reduced the recognition error to zero, which further supports its suitability. Compared to GPSR, these results are notable as DeepNAG generates new samples purely from random noise. Conversely, GPSR perturbs existing samples to generate new ones. This process leaves some



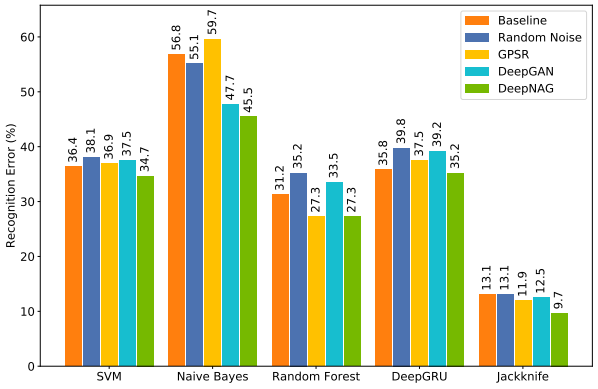
(a) JK2017 (Kinect) [44]



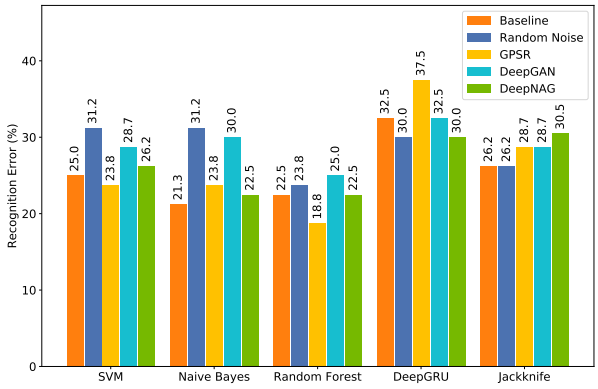
(b) JK2017 (Leap Motion) [44]



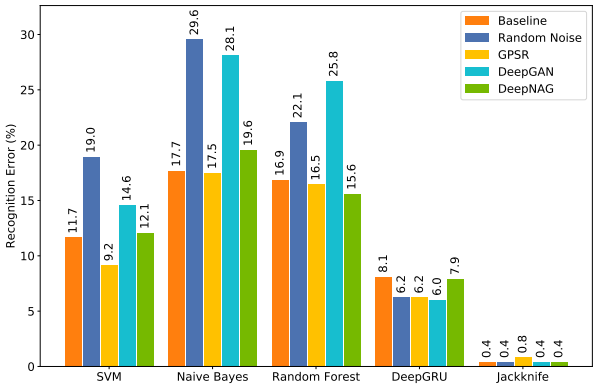
(c) UT-Kinect [3]



(d) MSR Action3D [163]



(e) SBU Kinect Interactions [129]



(f) $1-GDS [62]

Figure 6.5: Results of evaluation across six datasets (best viewed in color).

characteristics of the original gesture (*e.g.* bounding box size) largely unchanged, which benefits recognizers that rely on such features.

Figure 6.5 also shows cases wherein data augmentation seems harmful. In particular, we observe increased errors in almost all cases where data generation is used with multi-actor gestures (Figure 6.5e). This suggests that our generators may not be suitable for generating multi-actor gestures, which we confirmed by visual inspection. In some cases, both DeepGAN and DeepNAG confuse the main and the secondary actors, yielding malformed gestures. We intend to study the generation of such gestures in future work. We additionally inspected some of the generated samples of Figure 6.5f wherein our generators increased recognition errors. Most synthetic samples were in fact visually fine (as we show in our qualitative evaluations in Section 6.5.2) which suggests that the use of domain adaptation techniques may be helpful [30, 31, 164]. We explore this in Section 6.6.

During a visual inspection, we did not observe any mode-collapse issues with DeepNAG. We observed healthy variations across all gesture classes and datasets with minimal amounts of degenerate samples (except for the few cases noted above). Lastly, factors such as ease of training and training times compel the use of DeepNAG over DeepGAN as the former offers a significant reduction in training times. Training DeepGAN on a Tesla V100 GPU takes between 3-5 days depending on the dataset size, whereas DeepNAG takes around 3-7 *hours* under the same conditions, a speedup of 12–17$\times$.

6.5.2 Qualitative Evaluation

To evaluate the quality of the generated samples by DeepGAN and DeepNAG, we conducted a user study using the HYPE$_\infty$ benchmark [41]. Specifically, we replicated the protocol of HYPE$_\infty$ and used the Amazon Mechanical Turk platform to recruit participants and conduct our study.

Table 6.1: Generator scores aggregated over *dataset* and *recognizer*.

Dataset	**Generator Score**			**Recognizer**	**Generator Score**		
	GPSR	**DeepGAN**	**DeepNAG**		**GPSR**	**DeepGAN**	**DeepNAG**
JK2017 (Kinect) [44]	0	2	2	SVM	2	0	4
JK2017 (LeapMotion) [44]	0	1	4	Naïve Bayes	1	3	2
UT-Kinect [3]	0	1	2	Random Forest	2	1	3
MSR Action3D [163]	1	0	3	DeepGRU	0	1	3
SBU Kinect	2	0	0	Jackknife [44]	0	0	2
$1-GDS [62]	2	1	1				
Total Score	5	5	**12**		5	5	**14**

Our study design is as follows. Given a dataset $\mathbb{D}$ and a generator G we train the generator to convergence. For every gesture class, we synthesize as many fake samples as the real samples in $\mathbb{D}$ using the trained model. Thus if $\mathbb{D}$ contains N samples in total, we produce N_G samples from the trained model. This way, the generated sets will have the same number of samples from each class as $\mathbb{D}$. We then run the HYPE_∞ benchmark: we randomly sample 50 fake gestures and 50 real gestures from the pool of all available gesture sequences, while ensuring to maintain the ratio of different gesture classes in each set.

To conduct the study, we recruit 30 participants on Amazon Mechanical Turk. Participants begin by studying the purpose of the study and answering some demographic questions as detailed in Table 6.2. We then randomly show them each of the 100 samples and ask them to indicate whether they think a given sample is produced by a human or computer-generated (see Figure 6.6). Every gesture sequence is drawn in the form of a looping *gif* animation with a framerate of 32. Between each animation loop, we display a countdown with a duration of 0.25 seconds. This was inspired by [41] and was done to avoid confusing participants who may be unaware that they are watching an infinitely-looping animation.

Table 6.2: Pre-study questionnaire. Except for *age*, all other questions are multiple-choice.

Q1	What is your gender?
Q2	What is the highest level of education that you have completed?
Q3	What is your age?
Q4	Do you play video games?
Q5	How many hours per day? *(only if the participant plays video games)*

Participants are given an infinite amount of time to respond to each question. Similar to the HYPE_{∞} benchmark, we reveal the correct answer to the participant upon submitting a response to every question. Every participant is allowed to participate in our study only one time, thus ensuring a between-subject design across different datasets and generators. Once the study concludes, we reveal the overall accuracy of the participant in our task and pay them $2 for their time [41].

When posting our study on the Mechanical Turk platform, we created a list of criteria to ensure the selection of a pool of high-quality workers. First, participants must have an approval rating of at least 97% to participate in our study. An approval rating of 97% indicates that participants only had 3% of their work rejected by study requesters on the platform. This criterion filters out workers whose work is frequently rejected possibly due to the low quality of the work. Our next participation requirement is that workers must have completed at least 5000 studies. This criterion filters out participants who may have high approval ratings because they recently joined the platform. Lastly, participants must be *Mechanical Turk Masters* to be eligible to participate in our study. Amazon uses proprietary criteria to grant top-performing workers this qualification. Although the exact criteria are not publicly disclosed, Amazon claims they continuously monitor the performance of master workers across different user studies on the platform to ensure consistent performance[5].

[5]Details available at *https://www.mturk.com/worker/help*

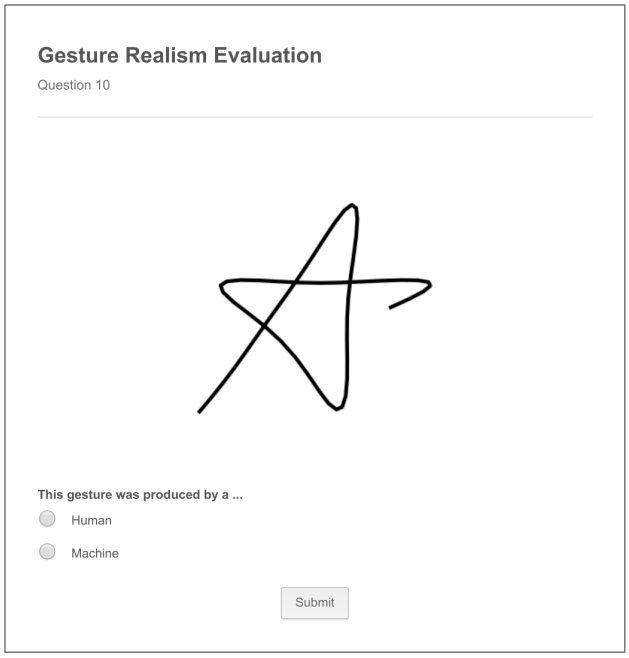


Figure 6.6: The interface of our user study application. Participants are shown the gesture animation and are asked to select either "human" or "machine". Once "submit" is clicked, the correct answer is revealed and the next gesture is displayed.

Our study consisted of evaluating each of DeepGAN and DeepNAG on three datasets covering different gesture modalities: Kinect (JK-2017 [44]), Leap Motion (JK-2017 [44]) and Pen gestures ($1-GDS [62]). Thus our *generator* factor has two levels and our *dataset* factor has three levels, yielding a total of six experiments.

6.5.2.1 Results and Discussion

We recruited 180 participants with an average age of 41 years (σ=10.8). Figure 6.7 depicts the demographics of our participants. A majority of our participants indicated that they played video games. Those who did, played an average of 2.2 hours per day (σ=1.8). Across all tasks, participants spent an average of 12.3 minutes (σ=3.8), and each question was answered in 7.4 seconds on

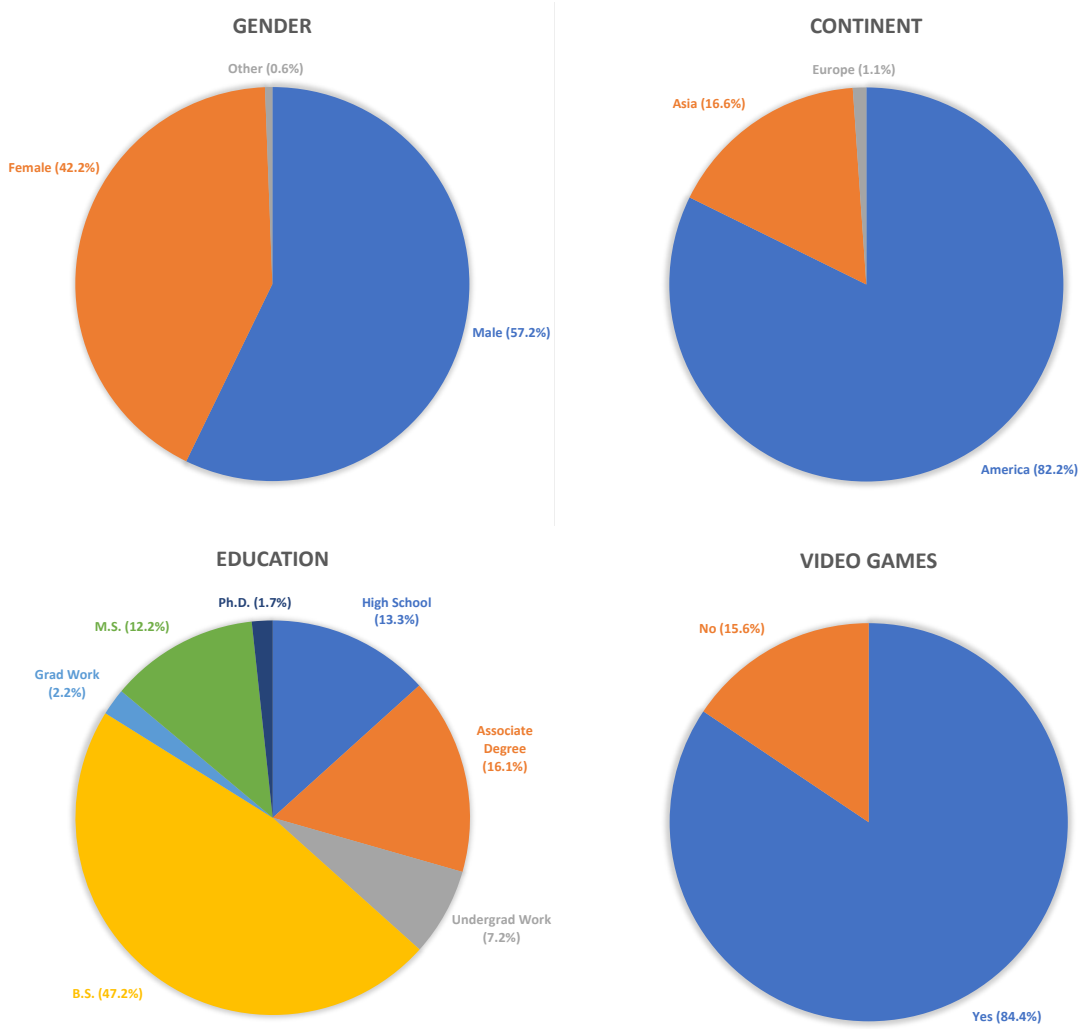


Figure 6.7: Demographic information of our user study participants.

average (σ=2.3). Considering a payment of $2 per study, our participants were compensated well above the minimum wage specified by the federal guidelines ($7.25/hr at the time of this writing).

Table 6.3 presents the results of our user study. In all experiments, we observe higher HYPE_{∞} scores for DeepNAG compared to DeepGAN. Unpaired t-tests confirm that the difference is significant in all experiments: $t(58)$=3.3, p=0.001 (JK2017-Kinect [44]), $t(58)$=12.4, p <0.001 (JK2017-Leap Motion [44]) and $t(58)$=2.8, p=0.006 ($1-GDS [62]).

Table 6.3: Amazon Mechanical Turk user study results. Reported values are percentages (averaged over 30 participants). The top performing model (with statistical significance) is boldfaced.

Dataset	Generator	HYPE_∞	Std.	Fake Errors	Real Errors
JK2017 (Kinect) [44]	**DeepNAG**	48.1	8.8	53.9	42.3
	DeepGAN	38.4	12.9	44.3	32.5
JK2017 (LeapMotion) [44]	**DeepNAG**	51.0	4.3	56.1	45.8
	DeepGAN	22.7	11.4	23.9	21.4
$1-GDS [62]	**DeepNAG**	50.0	6.7	56.3	43.7
	DeepGAN	44.4	8.3	49.5	39.3

Focusing on the results with JK2017 (Leap Motion) [44] dataset, we observe a large HYPE_∞ score gap between the two generators. Notably, DeepNAG achieves hyper-realism on this dataset: its fake samples look more realistic to humans than the real ones. These results correlate well with those in Section 6.5.1.1: on the Leap Motion dataset, DeepNAG significantly outperformed DeepGAN in reducing the recognition error (Table 6.1) and in some cases, DeepNAG reduced the recognition error to zero yielding perfect recognition accuracies (Figure 6.5).

Similar to [41], we report a breakdown of the error on the real and fake samples. We observe higher fake errors with DeepNAG in all cases. Additionally, real and fake errors track each other in every case, in line with Zhou *et al.*'s observation [41]. This indicates participants become more confused when fake samples are particularly hard to distinguish from the real ones.

To investigate whether there is an association between playing video games and distinguishing between real and fake samples, we performed a multiple regression analysis using *dataset*, *generator* and *play video games* as predictors. The results show that there is no statistically significant association between playing video games and *accuracy* when controlled for *dataset* and *generator* $\big($ coeff=0.01, p=0.51, CI (95%)=(-0.029, 0.058) $\big)$.

6.6 Experiments with Domain Adaptation: DeepCycleGAN

As can be seen in Figure 6.5, the use of data augmentation increases the recognition error in a small number of cases. To better understand this behavior, we manually inspected some of the samples. In the case of SBU Kinect Interactions [129] dataset, the increased recognition errors were due to the poor quality of the generated samples. In other cases, the synthetic samples were visually fine. In fact, as we showed in Section 6.5.2 the synthetic samples produced by DeepNAG on the $1-GDS [62] dataset were able to successfully deceive human evaluators. This observation motivated us to experiment with domain adaptation techniques and determine whether the observed behavior is due to the domain shift effect [29, 30].

As mentioned in Section 2.4.1.3, GANs can be used for mapping samples between domains. In this section, we discuss the design and evaluation of a novel recurrent CycleGAN model to map the synthetic samples of DeepNAG to the real sample domain. We evaluate the benefit of such mapping in data augmentation for improved gesture recognition. To our knowledge, this is the first time a recurrent gesture generator is trained in a CycleGAN framework towards domain adaptation.

6.6.1 Model Architecture

To perform domain mapping we require a GAN-based gesture generator. As discussed in Section 2.4.1.3, CycleGAN's loss function is agnostic to the underlying GANs which are trained for domain adaptation. As such, the natural choice for the two required GAN models is DeepGAN. Henceforth, we refer to our domain adaptation model as *DeepCycleGAN* which consists of two DeepGAN models trained using a CycleGAN framework as described in Section 2.4.1.3. The model signified as $\text{DeepGAN}_{\text{real}}$ uses the generator G to map synthetic samples to the real domain, whereas $\text{DeepGAN}_{\text{fake}}$ uses the generator F to map the real samples to the synthetic domain.

Table 6.4: Evaluating the recognition errors when performing domain adaptation. All reported values are percentages.

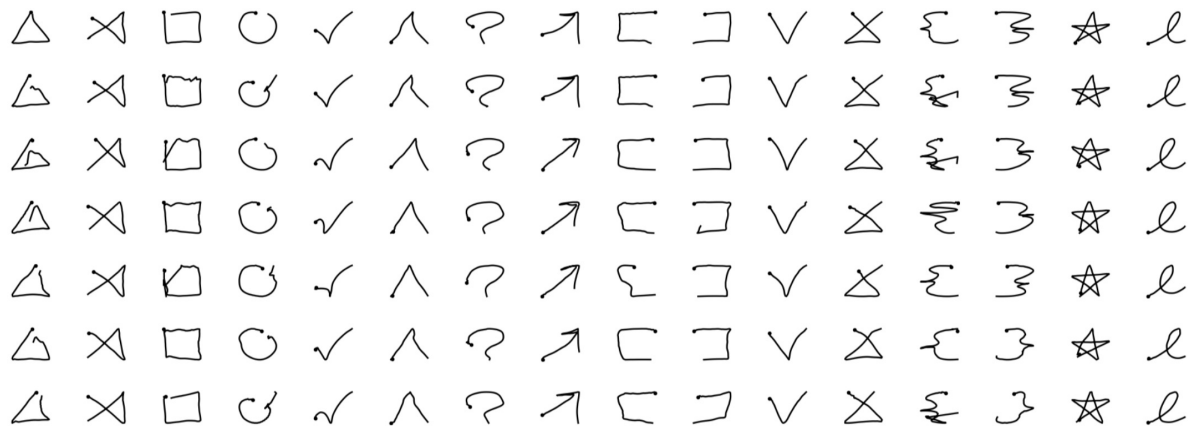
Recognizer	**Recog. Error with Synthetic Generator**		
	None	**DeepNAG**	**DeepNAG+DeepCycleGAN**
SVM	11.7	12.1	8.5
Naïve Bayes	17.7	19.6	20.4
Random Forest	16.9	15.6	17.7
DeepGRU	8.1	7.9	9.1
Jackknife	0.4	0.4	0.4

6.6.2 Training Loss Function

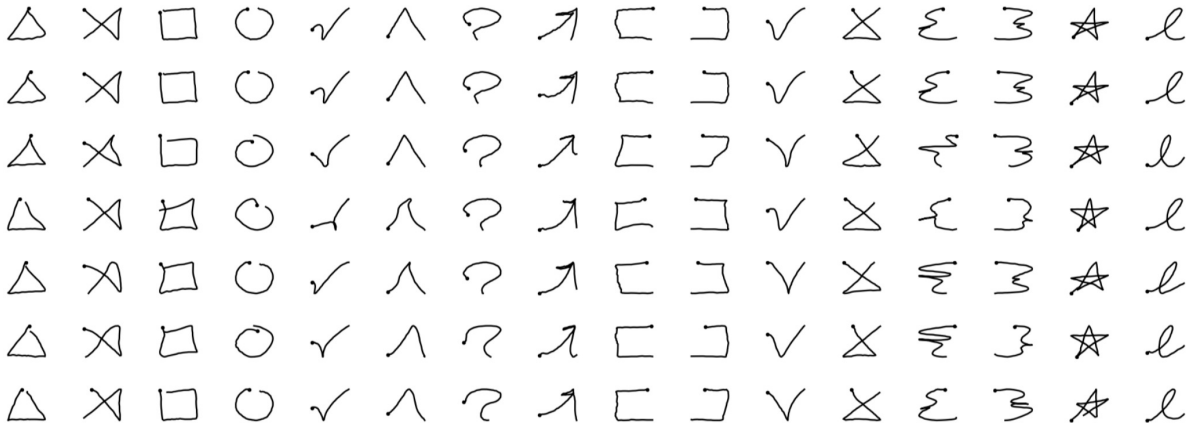
We use the following loss function to train DeepCycleGAN which resembles the original CycleGAN loss function [31]. We found the use of the identity loss term greatly beneficial for our problem domain. In our experiments we set λ_1=10 and λ_2=0.5 which we established experimentally.

$$\mathscr{L}_{\text{DeepCycleGAN}} = \mathscr{L}_{\text{DeepGAN}_{\text{real}}} + \mathscr{L}_{\text{DeepGAN}_{\text{fake}}} + \lambda_1 \mathscr{L}_{\text{cyc}}\big(G, F\big) + \lambda_2 \mathscr{L}_{\text{identity}}\big(G, F\big) \tag{6.7}$$

Although we trained DeepGAN with the WGAN-GP loss function for all of our experiments thus far, we observed poor training convergence and long training times when we used the improved Wasserstein loss function to train DeepCycleGAN's mappers. After some experimentation, we found the least squares GAN's (LSGAN) objective function [165] to significantly improve the training process and the quality of the generated samples.



(a) Synthetic samples mapped to the real domain



(b) Real samples mapped to the synthetic domain

Figure 6.8: The results of mapping samples between the real and the synthetic domains using DeepCycleGAN on $1-GDS [62] dataset.

6.6.3 Evaluation

To determine whether domain adaptation using DeepCycleGAN is beneficial we performed followup experiments with the DeepNAG model trained on $1-GDS [62]. We took the trained DeepNAG models of Section 6.5.1.1 and generated an entire dataset of synthetic samples $\mathbb{D}'$ and trained DeepCycleGAN to perform domain mapping between $\mathbb{D} = \{\$1\text{-GDS [62]}\}$ and $\mathbb{D}'$.

Table 6.4 summarizes the results of our experiments. Based on these results, DeepCycleGAN helped in reducing the recognition errors in only one case. In most other cases, using DeepCycleGAN actually increased recognition errors. To better understand these results we visually examined some synthetic samples (see Figure 6.8). We see that many of the mapped samples are noisy and demonstrate obvious defects. We hypothesize that the added noise on the mapped samples is potentially what improved the results for the SVM recognizer in Table 6.4.

In summary, the results in Table 6.4 and Figure 6.8 suggest that the DeepCycleGAN model is not functioning as expected. Perhaps one could achieve better results by a more careful design of the model's architecture. Given that such a design is out of the scope of this work, we leave a detailed study of this subject to future work.

6.7 Experiments with Variational Autoencoders: VAEG

Thus far, we have shown that training our proposed RNN-based gesture generator using our novel loss function outperforms the same generator that is trained with the improved WGAN loss in a GAN training setting. In this section we intend to investigate whether our generator can be trained in a variational autoencoder (VAE) setting.

As mentioned in Section 2.4.2, VAEs consist of encoder and decoder networks. Once training is complete, the decoder network can generate new synthetic samples which implies that the decoder acts as the synthetic generator which means it is possible to use DeepNAG's generator as the decoder in a VAE. We discuss the choice of encoder architecture shortly. We refer to our generator trained in a VAE framework as *VAEG* henceforth.

The first step in training VAEG is to define the training loss function. Recall from Section 2.4.1 that

the VAE objective is defined as follows and consists of reconstruction and regularization terms:

$$\mathscr{L}(\theta, \phi; \mathbf{x}, \mathbf{z}_{\hat{c}}) \;=\; \underbrace{-\mathbb{E}_{q_\phi(\mathbf{z}_{\hat{c}}|\mathbf{x})}\big[\log p_\theta(\mathbf{x}|\mathbf{z}_{\hat{c}})\big]}_{\text{reconstruction}} \;+\; \underbrace{D_{KL}\big(q_\phi(\mathbf{z}_{\hat{c}}|\mathbf{x}) \;||\; p(\mathbf{z}_{\hat{c}})\big)}_{\text{regularization}} \tag{6.8}$$

where $\mathbf{z}_{\hat{c}}$ is the latent vector conditioned on the class label c. Note that to adopt this loss function for gesture generation, the regularization term can be used without modifications, as it simply ensures that the learned latent space follows the standard normal distribution. The reconstruction term, however, is domain-specific. To our knowledge, no reconstruction loss term for generating gestures has been previously discussed in the literature. Recall that the reconstruction term ensures that the output of the decoder (generator) closely resembles the input data. Conveniently, a differentiable measure that can be used for this purpose is sDTW. Thus, we propose the following loss function as the objective to minimize during the training VAEG:

$$\mathscr{L}(\theta, \phi; \mathbf{x}, \mathbf{z}_{\hat{c}}) \;=\; \text{sDTW}\big(\mathbf{x}, G_\theta(\mathbf{z}_{\hat{c}}); f\big) \;+\; D_{KL}\big(q_\phi(\mathbf{z}_{\hat{c}}|\mathbf{x}) \;||\; p(\mathbf{z}_{\hat{c}})\big) \tag{6.9}$$

where f is sDTW's internal cost function. In simple terms, we define the reconstruction error as the sDTW dissimilarity between the input data, and the output of the generator. Given that $p(\mathbf{z}) = \mathcal{N}(0, 1)$, minimizing Equation 6.9 aims to decrease the reconstruction error and force the learned latent space to follow the standard normal distribution.

We now discuss the choice of the encoder network's architecture. We started with the uDeepGRU model as the encoder. This way, the VAEG model closely resembles that of DeepGAN's, yielding the overall architecture depicted in Figure 6.9. After running some preliminary experiments, we observed that the choice of the encoder architecture, as well as the overall model choices had

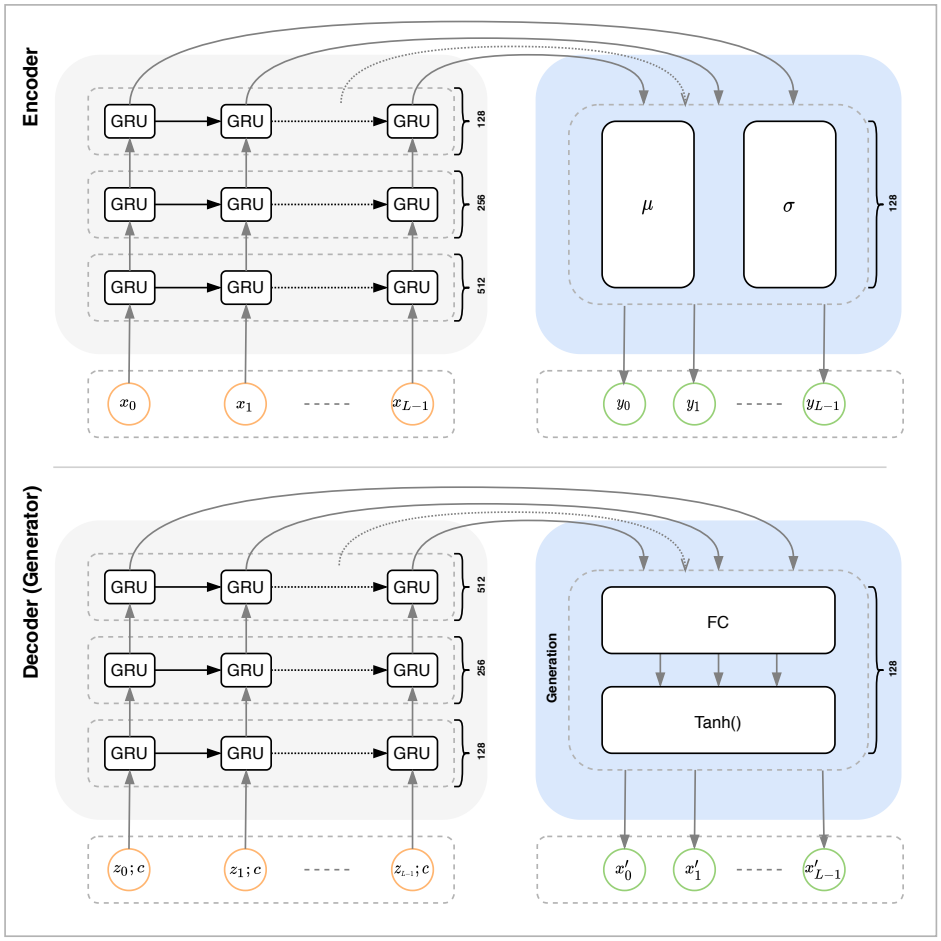
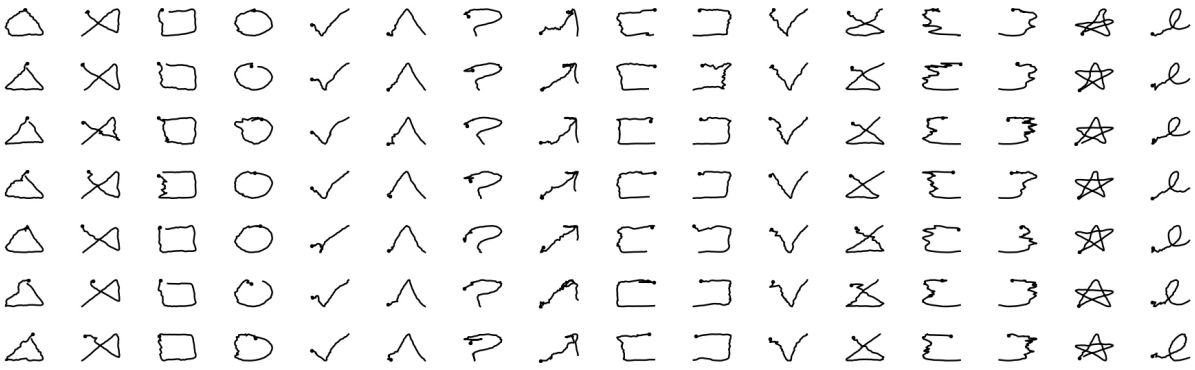


Figure 6.9: Our proposed VAE model for gesture generation dubbed *VAEG*. The encoder network is based on the uDeepGRU model (see Section 5.3). The generator is the same as DeepGAN/DeepNAG (see Section 6.2)

very little impact on the model's performance. Further, adding additional fully-connected layers in either the encoder or the decoder also made little difference in the produced results, in line with what Bowman *et al*. [166] observed. Although training plots showed a steady decrease of the loss value and convergence, the generated results lacked quality and diversity. Most gesture trajectories were rather noisy. We observed this trend regardless of the architecture of encoder or decoder networks, hyperparameter settings, the choice of f (we tried ED and COS) or even the gesture dataset. We additionally experimented with alternative reconstruction terms. Specifically,



(a) VAEG (samples)

(b) VAEG (overlays)

Figure 6.10: Some synthetic gestures produced by VAEG trained on $1-GDS [62] dataset. Note that most samples are noisy and lack visual quality. Overlayed rendering of synthetic samples from our VAE model (b – top), and real samples (b – bottom). Each overlay consists of 16 samples per class. Note the lack of variety in the synthetic results when compared to the real samples.

we experimented with the mean squared error (MSE) of both Euclidean distance as well as the cosine similarity of gesture paths ($\vec{\mathbf{x}}$) between the input and reconstructed samples. These alternate formulations performed worse than our sDTW-based reconstruction term.

Some samples produced by VAEG when trained on the $1-GDS dataset [62], along with overlayed samples for each of the real and synthetic data are depicted in Figure 6.10. These results show that the produced samples lack sufficient diversity when compared to the real samples. Given the visual quality of the results, we hypothesize that the VAE framework is unfit for training our generator to produce good synthetic samples.

6.8 Conclusion

We discussed modality-agnostic gesture generation with recurrent neural networks. We first presented DeepGAN, our GAN model for synthetic gesture generation across various datasets and gesture modalities. To reduce the training complexity, we formulated a novel loss function based on the dynamic time warping (DTW) algorithm and the average Hausdorff distance. Our loss function obviated the need for a separate discriminator network, yielded high-quality results, and led to 12–17$\times$ faster training. We called this approach DeepNAG and applied it towards data augmentation for improved gesture recognition. In our evaluations, DeepNAG outperformed other synthetic gesture generators (including DeepGAN) across various datasets and recognizers. We also briefly discussed the use of domain adaptation techniques to avoid problems such as domain shift when training a recognizer using synthetic data. We discussed the design of DeepCycleGAN, a deep recurrent network that maps samples between real and synthetic data domains. Preliminary experiments with this model suggested that further investigation is needed before we can successfully use such models for domain adaptation.

To evaluate the quality of the generated samples using our generators, we then designed and conducted a user study on the Amazon Mechanical Turk platform using the HYPE_∞ benchmark. Our results showed that across three gesture datasets, DeepNAG's fake samples were more frequently confused with the real ones compared to DeepGAN's. On one of our studied datasets, DeepNAG achieved hyper-realism indicating a high degree of visual quality in the generated samples.

Lastly, we discussed the training of our generator in a VAE setting. We adapted VAE's loss function to the task of gesture generation by adopting a sDTW-based reconstruction term. These modifications led to the design of VAEG, our VAE-based gesture generator. Our experiments showed that a VAE framework is not suitable for training our gesture generator since the produced results lacked quality and diversity when compared to those produced by DeepGAN and DeepNAG.

CHAPTER 7: CONCLUSION AND FUTURE DIRECTIONS

In this dissertation, we discussed gesture generation and recognition using recurrent deep networks. We formally introduced the recognition as well as the generation problems, and highlighted our design philosophies in formulating a solution for each problem.

Our design philosophy is accessibility: proposed models should be simple, have a small number of trainable parameters, and reusable across different tasks, yet perform well on evaluation measures. With these goals in mind, we detailed our current progress.

We presented the relevant background information and reviewed some of the concepts and terminologies pertinent to the task at hand. We then examined the existing related work on gesture recognition and generation and elaborated on how this work differs from all of them. We then proceeded to discuss and evaluate all of our proposed methods. Here, we provide a brief summary of our methods and findings.

7.1 DeepGRU for Segmented Gesture Recognition

We introduced *DeepGRU* for segmented gesture recognition and showed that our small network was able to achieve state-of-the-art results on many popular datasets, spanning over a broad range of interaction modalities, such as hand or full-body gestures with diverse data representations, such as skeletal or raw accelerometer measurements. We provided a comprehensive evaluation of DeepGRU and compared it against other methods available in the literature.

Our evaluations showed that DeepGRU outperformed many state-of-the-art methods, while achieving competitive recognition performance in most other cases. We then evaluated our model's per-

formance in the absence of large amounts of training data. With as little as four training samples per gesture class, our method outperformed other methods, including the state-of-the-art on two publicly available datasets.

We additionally inspected the time it took to train DeepGRU without powerful hardware. On small training sets, our model can be trained in under 10 minutes, while achieving competitive accuracy. We ended our discussion of DeepGRU with an ablation study which showed the effects of each component on the final performance of the system.

7.2 uDeepGRU for Continuous Gesture Recognition

While segmented gesture recognition is important, many practical applications of recognizers involve continuous streams of data. As such, we next turned our focus to the task of continuous gesture recognition. We discussed *uDeepGRU*, an extension of DeepGRU designed for working with unsegmented data which outputs per-frame recognition labels. The design of this model was guided by the ablation study that we conducted on DeepGRU.

We discussed the initial design of this model and a suitable loss function that worked well on continuous recognition tasks. Using this model, we participated in the SHREC 2019 Online Gesture Recognition challenge and took first place. We discussed the results of the competition, along with a comparison against other participating methods. Guided by the lessons learned while submitting our method, we worked towards improving our model further.

Our improvements included some changes in the model's architecture, as well as adopting *focal loss* as our training objective. This loss function, which was recently introduced to overcome imbalanced training sets, yielded improvements in the recognition accuracy across the board. We evaluated our improved uDeepGRU model on four publicly available datasets and showed that this

model not only outperformed its original counterpart, but was also able to outperform the sliding window version of DeepGRU.

7.3 DeepGAN and DeepNAG for Synthetic Gesture Generation

After discussing gesture recognition, we turned our focus to the problem of gesture generation to overcome training data shortage. To this end, we presented *DeepGAN* and *DeepNAG*, two novel extensions of DeepGRU which were designed for gesture generation tasks.

DeepGAN tackled the problem of gesture generation using a GAN-style training framework. The training process involved optimizing an adversarial loss function over two networks: the discriminator which aimed to determine whether a given example was real or fake, and a generator network whose goal was to fool the discriminator. The discriminator network's architecture was a slightly modified version of uDeepGRU, whereas the generator network resembled the flipped version of the discriminator. This architecture promoted simplicity and was found to work quite well in practice. However, training a GAN model is generally hard due to the need for training two networks at the same time.

With the goal of simplifying the training process, we introduced DeepNAG: a single-network solution capable of generating realistic fake gesture examples. We described our formulation of a single loss measure, which was based on the average Hausdorff distance and a differentiable formulation of the dynamic time warping algorithm dubbed *sDTW*. Using our loss function, one could directly train the generator network without needing a separate discriminator network. This, not only simplified the training process, but also resulted in training speedups of up to $17\times$, in line with our design philosophy of accessibility. We evaluated both of our proposed models from two aspects: improvements in gesture recognition through data augmentation (quantitative), and the

perceived realism of our produced samples by human evaluators in a user study (qualitative).

Our quantitative evaluations consisted of augmenting the training set of five commonly used gesture recognizers via four synthetic gesture generation methods, including DeepGAN and DeepNAG. The results showed that in many cases, the studied recognizers benefited from the additional training data produced by DeepNAG, more so than other synthetic generators. In a few cases, however, the use of synthetic data hurt the recognition performance.

Our qualitative evaluations consisted of evaluating the perceived realism of our synthetic samples using the recently introduced HYPE_{∞} benchmark, a standard benchmark to qualitatively evaluate various generative models. To this end, we recruited 180 human participants on the Amazon Mechanical Turk platform. We randomly showed each participant a mixture of real as well as synthesized gesture sequences from either DeepGAN or DeepNAG, and asked them to specify whether they think those sequences were produced by humans or were computer-generated. Across all of our experiments, human evaluators confused DeepNAG's synthetic samples with the real samples most frequently.

We ended our discussion on synthetic gesture generation by providing a preliminary study of adapting our proposed generators towards domain adaptation, as well as gesture generation in a variational autoencoder (VAE) setting. We discussed *DeepCycleGAN*, our novel recurrent model for domain adaptation, and evaluated its efficacy in improving gesture recognition accuracy when DeepNAG's synthetic samples were additionally processed by it. Our early results did not indicate an immediate benefit in using DeepCycleGAN. We leave a detailed study of this to future work.

We lastly discussed *VAEG*, an adaptation of DeepGAN to gesture generation in a VAE setting. We discussed our proposed loss function to train VAEG. Our loss formulation was based on minimizing sDTW and was functional in practice. The quality of the generated samples, however, was not on par with our expectations. We leave a detailed study of VAEG to future work.

7.4 Future Work

We outline a few key areas that we believe can be further explored in future work. As previously discussed, both DeepGRU and uDeepGRU were designed to work with vector or skeleton data. We believe both methods can be easily extended to support other types of input data such as videos or spoken words. To support videos, we foresee two possible approaches. The first approach involves using high-level computer vision features such as optical flows to extract discriminant features that the model could use for recognition. The second approach involves using convolutional layers for feature extraction as the first few layers of the model. Prior work has demonstrated the effectiveness of various types of convolution-based models towards this [76, 97]. One potential pitfall of the second approach is the added complexity to our models which could result in a larger and possibly slower model.

The recognition of spoken words can also be done with the use of convolutional layers. One could use several convolutional layers as the first few layers of the model to extract relevant features from spoken words. These features can then be fed to the GRU layers to perform recognition.

With respect to skeleton-based gesture recognition, other forms of extracting features from such data can also be explored. Given the graph-like structure of skeletal data, we foresee that one could represent such data using a graph and leverage graph convolutional networks (GCN) [167] to extract features from each frame's skeleton. However, this form of feature extraction will be more complex than using a few recurrent layers for this purpose. As such, these approaches will likely result in a more complex model which may be slower to train.

Besides supporting additional types of input data, we believe uDeepGRU can be improved even further. As discussed in Chapter 5, we did not employ any post-processing methods to refine the outputs produced by uDeepGRU. Using such methods could increase the overall accuracy of uDeep-

GRU's predictions. Similar approaches can also be used to refine the outputs of sliding-window DeepGRU. One possibility here is designing a post-processor that only registers a gesture if that gesture is detected for at least N frames. This approach could reduce the amount of misdetections.

Addressing the limitations of DeepGAN and DeepNAG is another avenue for future research. Our evaluations indicate that producing multi-actor gestures was challenging for both models. Additionally, both models are designed to produce synthetic samples of a fixed length, a limitation that can be addressed by small modifications in the generator model. We foresee the need for outputting an *"end-of-gesture"* token by the generator if the goal is to synthesize gestures of different lengths. This is an approach similar to the one proposed by Zhang *et al*. [118].

The preliminary results from DeepCycleGAN and VAEG in Chapter 6 suggest that a more thorough design and evaluation of both methods is warranted. Although our proposed loss functions led to visually acceptable results, we think both models as well as their loss functions can be improved. Although we did not observe any immediate benefits in changing VAEG's architecture, we did not examine alternate architectures for DeepCycleGAN which we think could improve domain adaptation performance.

In closing, we believe both DeepGAN and DeepNAG can be used in other application domains besides gestures. No component in the architecture of either network is specific to gestural data. As such, adapting either model to other real-valued sequential data should be straightforward.

APPENDIX: IRB EXEMPTION LETTERS

UNIVERSITY OF CENTRAL FLORIDA

Institutional Review Board
FWA00000351
IRB00001138, IRB00012110
Office of Research
12201 Research Parkway
Orlando, FL 32826-3246

EXEMPTION DETERMINATION

June 15, 2020

Dear Mehran Maghoumi:

On 6/15/2020, the IRB determined the following submission to be human subjects research that is exempt from regulation:

Type of Review:	Initial Study, Exempt Category 2
Title:	Gesture Realism Evaluation through Amazon Mechanical Turk
Investigator:	Mehran Maghoumi
IRB ID:	STUDY00001890
Funding:	None
Grant ID:	None
Documents Reviewed:	• Demographics.jpg, Category: Survey / Questionnaire; • HRP-254-FORM Explanation of Research.pdf, Category: Consent Form; • images.zip, Category: Other; • Instructions-Kinect.jpg, Category: Survey / Questionnaire; • Instructions-LeapMotion.jpg, Category: Survey / Questionnaire; • Instructions-Pen.jpg, Category: Survey / Questionnaire; • MechanicanTurk-ad.png, Category: Recruitment Materials; • Question-template.jpg, Category: Survey / Questionnaire; • Request for Exemption, Category: IRB Protocol;

This determination applies only to the activities described in the IRB submission and does not apply should any changes be made. If changes are made, and there are questions about whether these changes affect the exempt status of the human research, please submit a modification request to the IRB. Guidance on submitting Modifications and Administrative Check-in are detailed in the Investigator Manual (HRP-103), which can be found by navigating to the IRB Library within the IRB system. When you have completed your research, please submit a Study Closure request so that IRB records will be accurate.

Due to current COVID-19 restrictions, in-person research is not permitted to begin until you receive further correspondence from the Office of Research stating that the restrictions have been lifted.

If you have any questions, please contact the UCF IRB at 407-823-2901 or irb@ucf.edu. Please include your project title and IRB number in all correspondence with this office.

Sincerely,

Adrienne Showman
Designated Reviewer

UNIVERSITY OF CENTRAL FLORIDA

Institutional Review Board
FWA00000351
IRB00001138, IRB00012110
Office of Research
12201 Research Parkway
Orlando, FL 32826-3246

EXEMPTION DETERMINATION

August 3, 2020

Dear Mehran Maghoumi:

On 8/3/2020, the IRB determined the following submission to be human subjects research that is exempt from regulation:

Type of Review:	Modification / Update, Exempt Category 3
Title:	Gesture Realism Evaluation through Amazon Mechanical Turk
Investigator:	Mehran Maghoumi
IRB ID:	MOD00001148
Funding:	None
Grant ID:	None
Documents Reviewed:	• HRP-254-FORM Explanation of Research.pdf, Category: Consent Form; • MechanicalTurk-ad.png, Category: Recruitment Materials; • Request for Exemption, Category: IRB Protocol;

This determination applies only to the activities described in the IRB submission and does not apply should any changes be made. If changes are made, and there are questions about whether these changes affect the exempt status of the human research, please submit a modification request to the IRB. Guidance on submitting Modifications and Administrative Check-in are detailed in the Investigator Manual (HRP-103), which can be found by navigating to the IRB Library within the IRB system. When you have completed your research, please submit a Study Closure request so that IRB records will be accurate.

If you have any questions, please contact the UCF IRB at 407-823-2901 or irb@ucf.edu. Please include your project title and IRB number in all correspondence with this office.

Sincerely,

Adrienne Showman
Designated Reviewer

LIST OF REFERENCES

[1] F. M. Caputo, S. Burato, G. Pavan, T. Voillemin, H. Wannous, J. P. Vandeborre, M. Maghoumi, E. M. Taranta II, A. Razmjoo, J. J. LaViola Jr., F. Manganaro, S. Pini, G. Borghi, R. Vezzani, R. Cucchiara, H. Nguyen, M. T. Tran, and A. Giachetti. Online Gesture Recognition. In *Eurographics Workshop on 3D Object Retrieval*, 2019.

[2] Sergio Escalera, Xavier Baró, Jordi Gonzàlez, Miguel A. Bautista, Meysam Madadi, Miguel Reyes, Víctor Ponce-López, Hugo J. Escalante, Jamie Shotton, and Isabelle Guyon. Chalearn looking at people challenge 2014: Dataset and results. In *ECCV 2014 Workshops*, pages 459–473, 2015.

[3] L. Xia, C.C. Chen, and JK Aggarwal. View invariant human action recognition using histograms of 3d joints. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on*, pages 20–27. IEEE, 2012.

[4] Quentin De Smedt, Hazem Wannous, Jean-Philippe Vandeborre, Joris Guerry, Bertrand Le Saux, and David Filliat. Shrec'17 track: 3d hand gesture recognition using a depth and skeletal dataset. In *10th Eurographics Workshop on 3D Object Retrieval*, 2017.

[5] Joseph J LaViola. 3d gestural interaction: The state of the field. *International Scholarly Research Notices*, 2013, 2013.

[6] Daniel Weinland, Remi Ronfard, and Edmond Boyer. A survey of vision-based methods for action representation, segmentation and recognition. *Comput. Vis. Image Underst.*, 115(2):224–241, February 2011.

[7] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[8] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[9] Kyunghyun Cho, Bart van Merriënboer, Çağlar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics.

[10] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

[11] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[12] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *Proc. ICLR*, 2015.

[13] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015.

[14] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. *CoRR*, abs/1502.03044, 2015.

[15] Sarah Adel Bargal, Andrea Zunino, Donghyun Kim, Jianming Zhang, Vittorio Murino, and

Stan Sclaroff. Excitation backprop for rnns. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[16] Fabien Baradel, Christian Wolf, and Julien Mille. Pose-conditioned spatio-temporal attention for human action recognition. *CoRR*, abs/1703.10106, 2017.

[17] Fabien Baradel, Christian Wolf, Julien Mille, and Graham W. Taylor. Glimpse clouds: Human activity recognition from unstructured feature points. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[18] Z. Fan, X. Zhao, T. Lin, and H. Su. Attention based multi-view re-observation fusion network for skeletal action recognition. *IEEE Transactions on Multimedia*, pages 1–1, 2018.

[19] J. Liu, G. Wang, L. Duan, K. Abdiyeva, and A. C. Kot. Skeleton-based human action recognition with global context-aware attention lstm networks. *IEEE Transactions on Image Processing*, 27(4):1586–1599, April 2018.

[20] Sijie Song, Cuiling Lan, Junliang Xing, Wenjun Zeng, and Jiaying Liu. An end-to-end spatio-temporal attention model for human action recognition from skeleton data. In *AAAI*, volume 1, pages 4263–4270, 2017.

[21] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[22] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *CoRR*, abs/1710.10196, 2017.

[23] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *CoRR*, abs/1411.1784, 2014.

[24] Cristobal Esteban, Stephanie L. Hyland, and Gunnar Ratsch. Real-valued (medical) time series generation with recurrent conditional gans, 2017.

[25] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 2234–2242. Curran Associates Inc., 2016.

[26] Lars Mescheder, Sebastian Nowozin, and Andreas Geiger. Which training methods for gans do actually converge? In *International Conference on Machine Learning (ICML)*, 2018.

[27] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *Proceedings of the 34nd International Conference on Machine Learning, ICML 2017, Sydney, Australia*, 2017.

[28] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein GANs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems NIPS'17*, 2017.

[29] Konstantinos Bousmalis, Nathan Silberman, David Dohan, Dumitru Erhan, and Dilip Krishnan. Unsupervised pixel-level domain adaptation with generative adversarial networks, 2016.

[30] Cycada: Cycle consistent adversarial domain adaptation. In *International Conference on Machine Learning (ICML)*, 2018.

[31] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, 2017.

[32] Sheng-Wei Huang, Che-Tsung Lin, Shu-Ping Chen, Yen-Yi Wu, Po-Hao Hsu, and Shang-Hong Lai. Auggan: Cross domain adaptation with gan-based data augmentation. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision – ECCV 2018*, pages 731–744, Cham, 2018. Springer International Publishing.

[33] Aayush Bansal, Shugao Ma, Deva Ramanan, and Yaser Sheikh. Recycle-gan: Unsupervised video retargeting. In *ECCV*, 2018.

[34] Dana H. Ballard. Modular learning in neural networks. In *Proceedings of the Sixth National Conference on Artificial Intelligence - Volume 1*, AAAI'87, page 279–284. AAAI Press, 1987.

[35] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[36] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2013.

[37] Christopher P. Burgess, Irina Higgins, Arka Pal, Loic Matthey, Nick Watters, Guillaume Desjardins, and Alexander Lerchner. Understanding disentangling in β-vae, 2018.

[38] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 3483–3491. Curran Associates, Inc., 2015.

[39] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in neural information processing systems*, pages 6626–6637, 2017.

[40] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Re-

thinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[41] Sharon Zhou, Mitchell Gordon, Ranjay Krishna, Austin Narcomey, Li F Fei-Fei, and Michael Bernstein. Hype: A benchmark for human eye perceptual evaluation of generative models. In *Advances in Neural Information Processing Systems*, pages 3449–3461, 2019.

[42] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(1):43–49, February 1978.

[43] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '12)*, 2012.

[44] Eugene M. Taranta II, Amirreza Samiei, Mehran Maghoumi, Pooya Khaloo, Corey R. Pittman, and Joseph J. LaViola Jr. Jackknife: A reliable recognizer with few samples and many modalities. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*, 2017.

[45] Eugene M. Taranta, II and Joseph J. LaViola, Jr. Penny pincher: A blazing fast, highly accurate $-family recognizer. In *Proceedings of the 41st Graphics Interface Conference (GI '15)*, 2015.

[46] Marco Cuturi and Mathieu Blondel. Soft-dtw: a differentiable loss function for time-series. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 894–903. JMLR. org, 2017.

[47] Ivan E Sutherland. Sketchpad a man-machine graphical communication system. *Simulation*, 2(5):R–3, 1964.

[48] Dean Rubine. Specifying gestures by example. In *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '91, pages 329–337, 1991.

[49] M. Devanne, H. Wannous, S. Berretti, P. Pala, M. Daoudi, and A. Del Bimbo. 3-d human action recognition by shape analysis of motion trajectories on riemannian manifold. *IEEE Transactions on Cybernetics*, 45(7):1340–1352, July 2015.

[50] Jorge Fernández-Ramírez, Andrés Álvarez-Meza, and Álvaro Orozco-Gutiérrez. Video-based human action recognition using kernel relevance analysis. In *Advances in Visual Computing*, pages 116–125, 2018.

[51] JF Hu, WS Zheng, J Lai, and J Zhang. Jointly learning heterogeneous features for rgb-d activity recognition. *IEEE transactions on pattern analysis and machine intelligence*, 39(11):2186–2200, 2017.

[52] Jian-Fang Hu, Wei-Shi Zheng, Lianyang Ma, Gang Wang, and Jianhuang Lai. Real-time rgb-d activity prediction by soft regression. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 280–296, Cham, 2016. Springer International Publishing.

[53] Rim Slama, Hazem Wannous, Mohamed Daoudi, and Anuj Srivastava. Accurate 3d action recognition using learning on the grassmann manifold. *Pattern Recogn.*, 48(2):556–567, February 2015.

[54] Michalis Vrigkas, Ermioni Mastora, Christophoros Nikou, and Ioannis A. Kakadiaris. Ro-

bust incremental hidden conditional random fields for human action recognition. In *Advances in Visual Computing*, 2018.

[55] Salman Cheema, Michael Hoffman, and Joseph J. LaViola. 3d gesture classification with linear acceleration and angular velocity sensing devices for video games. *Entertainment Computing*, 4(1):11 – 24, 2013.

[56] J. Weng, C. Weng, and J. Yuan. Spatio-temporal naive-bayes nearest-neighbor (st-nbnn) for skeleton-based action recognition. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 445–454, July 2017.

[57] O. Boiman, E. Shechtman, and M. Irani. In defense of nearest-neighbor based image classification. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, June 2008.

[58] R. Vemulapalli, F. Arrate, and R. Chellappa. Human action recognition by representing 3d skeletons as points in a lie group. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 588–595, June 2014.

[59] Pei Wang, Chunfeng Yuan, Weiming Hu, Bing Li, and Yanning Zhang. Graph based skeleton motion representation and similarity measurement for action recognition. In *European Conference on Computer Vision*, pages 370–385. Springer, 2016.

[60] X. Chen, H. Guo, G. Wang, and L. Zhang. Motion feature augmented recurrent neural network for skeleton-based dynamic hand gesture recognition. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 2881–2885, Sept 2017.

[61] G. Evangelidis, G. Singh, and R. Horaud. Skeletal quads: Human action recognition using joint quadruples. In *2014 22nd International Conference on Pattern Recognition*, pages 4513–4518, Aug 2014.

[62] Jacob O Wobbrock, Andrew D Wilson, and Yang Li. Gestures Without Libraries, Toolkits or Training: A $1 Recognizer for User Interface Prototypes. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*, UIST '07, 2007.

[63] Yang Li. Protractor: A Fast and Accurate Gesture Recognizer. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 2169–2172, New York, NY, USA, 2010. ACM.

[64] Lisa Anthony and Jacob O Wobbrock. A Lightweight Multistroke Recognizer for User Interface Prototypes. In *Proceedings of Graphics Interface 2010*, GI '10, pages 245–252, Toronto, Ont., Canada, Canada, 2010. Canadian Information Processing Society.

[65] Lisa Anthony and Jacob O Wobbrock. $N-protractor: A Fast and Accurate Multistroke Recognizer. In *Proceedings of Graphics Interface 2012*, GI '12, pages 117–120, Toronto, Ont., Canada, Canada, 2012. Canadian Information Processing Society.

[66] Radu-Daniel Vatavu, Lisa Anthony, and Jacob O Wobbrock. Gestures As Point Clouds: A $P Recognizer for User Interface Prototypes. In *Proceedings of the 14th ACM International Conference on Multimodal Interaction*, ICMI '12, pages 273–280, New York, NY, USA, 2012. ACM.

[67] Radu-Daniel Vatavu. Improving Gesture Recognition Accuracy on Touch Screens for Users with Low Vision. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, pages 4667–4679, New York, NY, USA, 2017. ACM.

[68] Radu-Daniel Vatavu, Lisa Anthony, and Jacob O Wobbrock. $ Q: a super-quick, articulation-invariant stroke-gesture recognizer for low-resource devices. In *Proceedings of the 20th International Conference on Human-Computer Interaction with Mobile Devices and Services*, page 23. ACM, 2018.

[69] J. Herold and T. F. Stahovich. The one cent recognizer: A fast, accurate, and easy-to-implement handwritten gesture recognition technique. In *Proceedings of the International Symposium on Sketch-Based Interfaces and Modeling*, SBIM '12, pages 39–46, Goslar Germany, Germany, 2012. Eurographics Association.

[70] Sven Kratz and Michael Rohs. The $3 recognizer: simple 3D gesture recognition on mobile devices. In *Proceedings of the 15th international conference on Intelligent user interfaces*, pages 419–420. ACM, 2010.

[71] Sven Kratz and Michael Rohs. Protractor3d: A closed-form solution to rotation-invariant 3d gestures. In *Proceedings of the 16th International Conference on Intelligent User Interfaces*, IUI '11, pages 371–374, New York, NY, USA, 2011. ACM.

[72] Jean Vanderdonckt, Paolo Roselli, and Jorge Luis Pérez-Medina. ! ftl, an articulation-invariant stroke gesture recognizer with controllable position, scale, and rotation invariances. In *Proceedings of the 2018 on International Conference on Multimodal Interaction*, pages 125–134. ACM, 2018.

[73] Fabio M Caputo, Pietro Prebianca, Alessandro Carcangiu, Lucio D Spano, and Andrea Giachetti. Comparing 3d trajectories for simple mid-air gesture recognition. *Computers & Graphics*, 73:17–25, 2018.

[74] D. Avola, M. Bernardi, L. Cinque, G. L. Foresti, and C. Massaroni. Exploiting recurrent neural networks and leap motion controller for the recognition of sign language and semaphoric hand gestures. *IEEE Transactions on Multimedia*, pages 1–1, 2018.

[75] Q. Ke, S. An, M. Bennamoun, F. Sohel, and F. Boussaid. Skeletonnet: Mining deep part features for 3-d action recognition. *IEEE Signal Processing Letters*, 24(6):731–735, June 2017.

[76] G. Devineau, F. Moutarde, W. Xi, and J. Yang. Deep learning for hand gesture recognition on skeletal data. In *2018 13th IEEE International Conference on Automatic Face Gesture Recognition (FG 2018)*, pages 106–113, May 2018.

[77] Amir Shahroudy, Jun Liu, Tian-Tsong Ng, and Gang Wang. Ntu rgb+d: A large scale dataset for 3d human activity analysis. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[78] P. Zhang, C. Lan, J. Xing, W. Zeng, J. Xue, and N. Zheng. View adaptive recurrent neural networks for high performance human action recognition from skeleton data. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2136–2145, Oct 2017.

[79] Jun Liu, Amir Shahroudy, Dong Xu, and Gang Wang. Spatio-temporal lstm with trust gates for 3d human action recognition. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 816–833, Cham, 2016. Springer International Publishing.

[80] I. Lee, D. Kim, S. Kang, and S. Lee. Ensemble deep learning for skeleton-based action recognition using temporal sliding lstm networks. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 1012–1020, Oct 2017.

[81] Juan C. Núñez, Ral Cabido, Juan J. Pantrigo, Antonio S. Montemayor, and Jos F. Vélez. Convolutional neural networks and long short-term memory for skeleton-based human activity and hand gesture recognition. *Pattern Recogn.*, 76(C):80–94, April 2018.

[82] Junwu Weng, Mengyuan Liu, Xudong Jiang, and Junsong Yuan. Deformable pose traversal convolution for 3d action and gesture recognition. In *Europen Conference on Computer Vision (ECCV)*, 2018.

[83] Corey R. Pittman and Joseph J. LaViola, Jr. Multiwave: Complex hand gesture recognition

using the doppler effect. In *Proceedings of the 43rd Graphics Interface Conference*, GI '17, pages 97–106, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2017. Canadian Human-Computer Communications Society.

[84] Vladimir I. Pavlovic, Rajeev Sharma, and Thomas S. Huang. Visual interpretation of hand gestures for human-computer interaction: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19:677–695, 1997.

[85] Sergio Escalera, Vassilis Athitsos, and Isabelle Guyon. *Challenges in Multi-modal Gesture Recognition*, pages 1–60. Springer International Publishing, Cham, 2017.

[86] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 282–289, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[87] Tian-Shu Wang, Heung-Yeung Shum, Ying-Qing Xu, and Nan-Ning Zheng. Unsupervised analysis of human gestures. In Heung-Yeung Shum, Mark Liao, and Shih-Fu Chang, editors, *Advances in Multimedia Information Processing — PCM 2001*, pages 174–181, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[88] Ying Yin and Randall Davis. Gesture spotting and recognition using salience detection and concatenated hidden markov models. In *Proceedings of the 15th ACM on International Conference on Multimodal Interaction*, ICMI '13, pages 489–494, New York, NY, USA, 2013. ACM.

[89] H. Yang, S. Sclaroff, and S. Lee. Sign language spotting with a threshold model based on conditional random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(7):1264–1277, July 2009.

[90] R. Oka. Spotting method for classification of real world data. *The Computer Journal*, 41(8):559–565, Jan 1998.

[91] J. Alon, V. Athitsos, Q. Yuan, and S. Sclaroff. A unified framework for gesture recognition and spatiotemporal gesture segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(9):1685–1699, Sep. 2009.

[92] Jingren Tang, Hong Cheng, Yang Zhao, and Hongliang Guo. Structured dynamic time warping for continuous hand trajectory gesture recognition. *Pattern Recognition*, 80:21 – 31, 2018.

[93] O Koller, O Zargaran, H Ney, and R Bowden. Deep sign: Hybrid cnn-hmm for continuous sign language recognition. In *The British Machine Vision Conference (BMVC) 2016*, September 2016.

[94] Zhaoyang Yang, Zhenmei Shi, Xiaoyong Shen, and Yu-Wing Tai. Sf-net: Structured feature network for continuous sign language recognition, 2019.

[95] Runpeng Cui, Hu Liu, and Changshui Zhang. Recurrent convolutional neural networks for continuous sign language recognition by staged optimization. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[96] P. Molchanov, S. Gupta, K. Kim, and K. Pulli. Multi-sensor system for driver's hand-gesture recognition. In *2015 11th IEEE International Conference and Workshops on Automatic Face and Gesture Recognition (FG)*, volume 1, pages 1–8, May 2015.

[97] P. Molchanov, S. Gupta, K. Kim, and J. Kautz. Hand gesture recognition with 3d convolutional neural networks. In *2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1–7, June 2015.

[98] P. Molchanov, X. Yang, S. Gupta, K. Kim, S. Tyree, and J. Kautz. Online detection and classification of dynamic hand gestures with recurrent 3d convolutional neural networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4207–4215, June 2016.

[99] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 369–376, New York, NY, USA, 2006. ACM.

[100] Guangming Zhu, Liang Zhang, Peiyi Shen, Juan Song, Syed Afaq Shah, and Mohammed Bennamoun. Continuous gesture segmentation and recognition using 3dcnn and convolutional lstm. *IEEE Transactions on Multimedia*, 2018.

[101] N. Dhingra and A. Kunz. Res3atn - deep 3d residual attention network for hand gesture recognition in videos. In *2019 International Conference on 3D Vision (3DV)*, pages 491–501, Sep. 2019.

[102] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-time human pose recognition in parts from single depth images. In *CVPR 2011*, pages 1297–1304, June 2011.

[103] Andreas Fischer, Muriel Visani, Van Cuong Kieu, and Ching Y. Suen. Generation of learning samples for historical handwriting recognition using image degradation. In *Proceedings of the 2Nd International Workshop on Historical Document Imaging and Processing*, HIP '13, pages 73–79, New York, NY, USA, 2013. ACM.

[104] Tamás Varga, Daniel Kilchhofer, and Horst Bunke. Template-based synthetic handwriting generation for the training of recognition systems. In *Proceedings of the 12th Conference of the International Graphonomics Society*, pages 206–211, 2005.

[105] Do-Hoon Lee and Hwan-Gue Cho. A new synthesizing method for handwriting korean scripts. *International Journal of Pattern Recognition and Artificial Intelligence*, 12(01):45–61, 1998.

[106] Javier Galbally, Julian Fierrez, Marcos Martinez-Diaz, and Javier Ortega-Garcia. Synthetic generation of handwritten signatures based on spectral analysis. In *SPIE Defense, Security, and Sensing*, pages 730629–730629. International Society for Optics and Photonics, 2009.

[107] Emilie Lundin, Håkan Kvarnström, and Erland Jonsson. A synthetic fraud data generation methodology. In *Proceedings of the 4th International Conference on Information and Communications Security*, ICICS '02, pages 265–277, London, UK, UK, 2002. Springer-Verlag.

[108] Eugene M Taranta II, Mehran Maghoumi, Corey R Pittman, and Joseph J LaViola Jr. A rapid prototyping approach to synthetic data generation for improved 2d gesture recognition. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, pages 873–885. ACM, 2016.

[109] Réjean Plamondon. A kinematic theory of rapid human movements. *Biological Cybernetics*, 72(4):295–307, Mar 1995.

[110] Réjean Plamondon and Moussa Djioua. A multi-level representation paradigm for handwriting stroke generation. *Human movement science*, 25(4):586–607, 2006.

[111] Luis A. Leiva, Daniel Martín-Albo, and Réjean Plamondon. Gestures À go go: Authoring synthetic human-like stroke gestures using the kinematic theory of rapid movements. *ACM Trans. Intell. Syst. Technol.*, 7(2):15:1–15:29, November 2015.

[112] Luis A. Leiva, Daniel Martín-Albo, and Réjean Plamondon. The kinematic theory produces human-like stroke gestures. *Interacting with Computers*, 29(4):552–565, July 2017.

[113] Kenny Davila, Stephanie Ludi, and Richard Zanibbi. Using off-line features and synthetic data for on-line handwritten math symbol recognition. In *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*, pages 323–328. IEEE, 2014.

[114] Ken Perlin. An Image Synthesizer. In *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '85, pages 287–296, New York, NY, USA, 1985. ACM.

[115] Alex Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013.

[116] Hao Tang, Wei Wang, Dan Xu, Yan Yan, and Nicu Sebe. GestureGAN for Hand Gesture-to-Gesture Translation in the Wild. In *Proceedings of the 26th ACM International Conference on Multimedia*, MM '18, pages 774–782. ACM, 2018.

[117] Ceyuan Yang, Zhe Wang, Xinge Zhu, Chen Huang, Jianping Shi, and Dahua Lin. Pose guided human video generation. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision – ECCV 2018*, pages 204–219, Cham, 2018. Springer International Publishing.

[118] X. Zhang, F. Yin, Y. Zhang, C. Liu, and Y. Bengio. Drawing and recognizing chinese characters with recurrent neural network. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4):849–862, April 2018.

[119] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, pages 2852–2858. AAAI Press, 2017.

[120] Jiaxian Guo, Sidi Lu, Han Cai, Weinan Zhang, Yong Yu, and Jun Wang. Long text gen-

eration via adversarial training with leaked information. *arXiv preprint arXiv:1709.08624*, 2017.

[121] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 3540–3549. JMLR.org, 2017.

[122] Kevin Lin, Dianqi Li, Xiaodong He, Zhengyou Zhang, and Ming-Ting Sun. Adversarial Ranking for Language Generation. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, 2017.

[123] Zhongliang Li, Tian Xia, Xingyu Lou, Kaihe Xu, Shaojun Wang, and Jing Xiao. Adversarial discrete sequence generation without explicit neuralnetworks as discriminators. In Kamalika Chaudhuri and Masashi Sugiyama, editors, *Proceedings of Machine Learning Research*, volume 89 of *Proceedings of Machine Learning Research*, pages 3089–3098. PMLR, 2019.

[124] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[125] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[126] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. pages 448–456, 2015.

[127] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[128] Quentin De Smedt, Hazem Wannous, and Jean-Philippe Vandeborre. Skeleton-based dynamic hand gesture recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 1–9, 2016.

[129] Kiwon Yun, Jean Honorio, Debaleena Chattopadhyay, Tamara L. Berg, and Dimitris Samaras. Two-person interaction detection using body-pose features and multiple instance learning. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on*. IEEE, 2012.

[130] C. Wang, Y. Wang, and A. L. Yuille. Mining 3d key-pose-motifs for action recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2639–2647, June 2016.

[131] R. Anirudh, P. Turaga, J. Su, and A. Srivastava. Elastic functional coding of human actions: From vector-fields to latent variables. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3147–3155, June 2015.

[132] Chunyu Wang, John Flynn, Yizhou Wang, and Alan L. Yuille. Recognizing actions in 3d using action-snippets and activated simplices. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pages 3604–3610. AAAI Press, 2016.

[133] Piotr Koniusz, Anoop Cherian, and Fatih Porikli. Tensor representations via kernel linearization for action recognition from 3d skeletons. In *European Conference on Computer Vision*, pages 37–53. Springer, 2016.

[134] Yansong Tang, Yi Tian, Jiwen Lu, Peiyang Li, and Jie Zhou. Deep progressive reinforcement learning for skeleton-based action recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[135] Yusuf Tas and Piotr Koniusz. Cnn-based action recognition and supervised domain adaptation on 3d body skeletons via kernel feature maps. In *BMVC*, 2018.

[136] Anoop Cherian, Suvrit Sra, Stephen Gould, and Richard Hartley. Non-linear temporal subspace representations for activity recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2197–2206, 2018.

[137] Diogo C Luvizon, David Picard, and Hedi Tabia. 2d/3d pose estimation and action recognition using multitask deep learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, 2018.

[138] A. Shahroudy, T. Ng, Y. Gong, and G. Wang. Deep multimodal feature analysis for action recognition in rgb+d videos. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(5):1045–1058, May 2018.

[139] F. Baradel, C. Wolf, and J. Mille. Human action recognition: Pose-based attention draws focus to hands. In *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, pages 604–613, Oct 2017.

[140] Yong Du, W. Wang, and L. Wang. Hierarchical recurrent neural network for skeleton based action recognition. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1110–1118, June 2015.

[141] Pichao Wang, Wanqing Li, Chuankun Li, and Yonghong Hou. Action recognition based on joint trajectory maps with convolutional neural networks. *Knowledge-Based Systems*, 158:43 – 53, 2018.

[142] H. Wang and L. Wang. Beyond joints: Learning representations from primitive geometries for skeleton-based action recognition and detection. *IEEE Transactions on Image Processing*, 27(9):4382–4394, Sept 2018.

[143] Qiuhong Ke, Mohammed Bennamoun, Senjian An, Ferdous Sohel, and Farid Boussaid. A new representation of skeleton sequences for 3d action recognition. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 4570–4579. IEEE, 2017.

[144] Mengyuan Liu, Hong Liu, and Chen Chen. Enhanced skeleton visualization for view invariant human action recognition. *Pattern Recogn.*, 68(C):346–362, August 2017.

[145] Shuchen Weng, Wenbo Li, Yi Zhang, and Siwei Lyu. Sts classification with dual-stream cnn. *arXiv preprint arXiv:1805.07740*, 2018.

[146] Quentin De Smedt, Hazem Wannous, and Jean-Philippe Vandeborre. 3d hand gesture recognition by analysing set-of-joints trajectories. In Hazem Wannous, Pietro Pala, Mohamed Daoudi, and Francisco Flórez-Revuelta, editors, *Understanding Human Activities Through 3D Sensors*, pages 86–97, Cham, 2018. Springer International Publishing.

[147] E. Ohn-Bar and M. M. Trivedi. Joint angles similarities and hog2 for action recognition. In *2013 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 465–470, June 2013.

[148] S. Y. Boulahia, E. Anquetil, F. Multon, and R. Kulpa. Dynamic hand gesture recognition based on 3d pattern assembled trajectories. In *2017 Seventh International Conference on Image Processing Theory, Tools and Applications (IPTA)*, pages 1–6, Nov 2017.

[149] Q. De Smedt, H. Wannous, and J. Vandeborre. Skeleton-based dynamic hand gesture recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1206–1214, June 2016.

[150] Y. Ji, G. Ye, and H. Cheng. Interactive body part contrast mining for human interaction recognition. In *2014 IEEE International Conference on Multimedia and Expo Workshops (ICMEW)*, pages 1–6, July 2014.

[151] Wentao Zhu, Cuiling Lan, Junliang Xing, Wenjun Zeng, Yanghao Li, Li Shen, and Xiaohui Xie. Co-occurrence feature learning for skeleton based action recognition using regularized deep lstm networks. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pages 3697–3703. AAAI Press, 2016.

[152] Sven Kratz and Michael Rohs. The $3 recognizer: Simple 3d gesture recognition on mobile devices. In *Proceedings of the 15th International Conference on Intelligent User Interfaces*, IUI '10, pages 419–420, New York, NY, USA, 2010. ACM.

[153] Jaime Lien, Nicholas Gillian, M. Emre Karagozler, Patrick Amihood, Carsten Schwesig, Erik Olson, Hakim Raja, and Ivan Poupyrev. Soli: Ubiquitous gesture sensing with millimeter wave radar. *ACM Trans. Graph.*, 35(4):142:1–142:19, July 2016.

[154] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. V-net: Fully convolutional neural networks for volumetric medical image segmentation. In *2016 Fourth International Conference on 3D Vision (3DV)*, pages 565–571. IEEE, 2016.

[155] Alex Graves. Supervised sequence labelling with recurrent neural networks. 2012.

[156] L. Liu, J. Shang, F. Xu, X. Ren, H. Gui, J. Peng, and J. Han. Empower sequence labeling with task-aware neural language model. In *AAAI*, 2018.

[157] T. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2999–3007, 2017.

[158] Jia Sheng. A study of adaboost in 3d gesture recognition. *Department of Computer Science, University of Toronto*, 2003.

[159] M. . Dubuisson and A. K. Jain. A modified hausdorff distance for object matching. In *Proceedings of 12th International Conference on Pattern Recognition*, volume 1, pages 566–568 vol.1, Oct 1994.

[160] Javier Ribera, David Güera, Yuhao Chen, and Edward J. Delp. Locating objects without bounding boxes. *Proceedings of the Computer Vision and Pattern Recognition (CVPR)*, June 2019. Long Beach, CA.

[161] H. Zhu, Z. Gu, H. Zhao, K. Chen, C. Li, and L. He. Developing a pattern discovery method in time series data and its gpu acceleration. *Big Data Mining and Analytics*, 1(4):266–283, 2018.

[162] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: A llvm-based python jit compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, LLVM '15. ACM, 2015.

[163] W. Li, Z. Zhang, and Z. Liu. Action recognition based on a bag of 3d points. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops*, 2010.

[164] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning, 2017.

[165] Xudong Mao, Qing Li, Haoran Xie, Raymond Y. K. Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks, 2016.

[166] Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space, 2015.

[167] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.